VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Implementace Continuous Integration prostředí v malé firmě

Implementation of Continuous Integration Enviroment in Small Company

2014 Zbyněk Ungermann

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Zbyněk Ungermann**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Implementace Continuous Integration prostředí v malé firmě
Implementation of Continuous Integration Enviroment in Small Company

Zásady pro vypracování:

Cílem diplomové práce je navrhnout a realizovat prostředí pro project management a continuous integration v reálné malé softwarové firmě. Důraz je kladen zejména na integraci projektů.

1. Navrhněte jednoduché procesy project managementu vývoje software realizované vybraným agilním přístupem. Navrhněte a integrujte nástroje pro jejich podporu.
2. Definujte potřebné KPI (key performance indicator) a navrhněte způsob podpory jejich sledování.
3. Prozkoumejte obecné postupy continuous integration, definujte nutné procesy s ohledem na malé firmy. Aplikujte nabyté znalosti pro projekty vyvíjené v prostředí .NET a DevExpress a jejich softwarovou podporu. Dále pak navrhněte a aplikujte způsob continuous integration a využití v prostředí JAVA.

Seznam doporučené odborné literatury:

Jez Humble, David Farley:"Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation", ISBN-13: 978-0321601919; Addison-Wesley Professional; 1 edition (August 6, 2010)

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Svatopluk Štolfa, Ph.D.**

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014

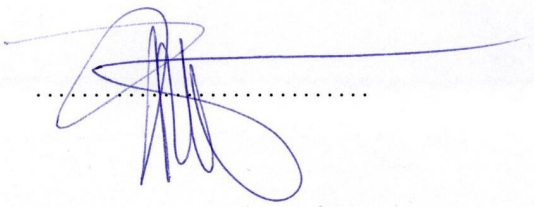
VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky OSTRAVA

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2014



.....

Rád bych poděkoval vedení a zaměstnancům společnosti ELVAC SOLUTIONS za podporu a otevřenost při implementaci nových procesů a trpělivost při překonávání úskalí, jenž tato implementace občas obnášela.

Abstrakt

Tato práce se zabývá implementací procesů pro podporu vývoje software použitím metod Continuous Integration a řízení projektů pomocí agilních metodik v malé společnosti, orientované na vývoj software na klíč. V této práci je popsán postup a vlastní implementace těchto procesů, jakož i výsledky, kterých bylo dosaženo. Dále se práce zbývá identifikací výkonnostních ukazatelů sloužících ke kontrole a hodnocení zde prezentovaných postupů.

Klíčová slova: Continuous Integration, výkonnostní ukazatele, KPI, projektové řízení, vývoj software, agilní metodiky, CruiseControl.NET

Abstract

This thesis deals with the topic of implementing processes supporting software development tasks using Continuous Integration methods and supporting project management tasks using Agile methods, implemented in a small sized company. The thesis describes the tasks and the actual implementation of these processes, as well as the results achieved. The approach used to identify performance metrics for project monitoring and evaluation of processes implemented is described in the thesis as well.

Keywords: Continuous Integration, performance indicators, KPI, project management, software development, agile, CruiseControl.NET

Seznam pojmů a zkratek

ACWP Actual Cost of Work Performed

BCWP Budgeted Cost of Work Performed

BCWS Budgeted Cost of Work Scheduled

branch jedná se o umístění zdrojových kódů řešení v SCM pro účely vývoje

build sestavení aplikace

check-in vložení části kódu do SCM

CI Continuous Integration

commit to samé jako check-in

CPI Cost Performance Index - ukazatel výkonosti projektu

CV Cost Variance - ukazatel výkonosti projektu

CVS Current Versions System - druh SCM

daily stand-up meeting Označení setkání projektového týmu ke konzultaci postupu prací a problémů pomocí metodiky SCRUM

FxCop Nástroj pro analýzu knihoven z pohledu doporučení Microsoft a .NET framewrok

KPI Key Performance Indicators - klíčové ukazatele kvality a výkonnosti na projektech

KPI Key Performance Indicators

Potentionally Shippable Product Označení výsledku iterace ve SCRUM terminilogii, značící že výsledkem každé iterace je hotová část systému, kterou může zákazník ihned použít.

Product Backlog Obsahuje plánované a požadované funkcionality software ve vývoji pomocí SCRUM

Product Owner Majitel (vlastník) projektu.

project steering group Řídící skupina projektu odpovědná za všechna rozhodnutí

release notes Release notes slouží jako seznam úkolů, funkcionalit a oprav, které byly vyřešeny v rámci jednoho vydání aplikace

SCM Source Configuration Management - správa zdrojových kódů

SCRUM Metodika pro vedené projektů a vývoje software za použití agilního přístupu.

SCRUM Master Role v projektovém řízení pomocí SCRUM. Odpovídá roli projektového vedoucího.

SPI Schedule Performance Index - ukazatel výkonosti projektu

StyleCop Nástroj pro analýzu zdrojového kódu z pohledu syntaxe, komentářů a dokumentace

SV Schedule Variance - ukazatel výkonosti projektu

SVN subversion - druh SCM

TDD Test Driven Development

trunk jedná se o hlavní větev zdrojových kódů řešení v SCM

Work Breakdown Structure Rozdělení prací do celků, částí a úkolů

XP eXtreme programming

Obsah

1 Úvod

Cílem této práce je implementace a podpora Continuous Integration procesů ve společnosti zabývající se vývojem software. Vlastní implementace těchto procesů proběhla ve společnosti ELVAC SOLUTIONS, kde jsem zaměstnán na pozici Continuous Integration Manager. Mým úkolem při nástupu do této společnosti bylo zlepšení kvality vývoje softwarových produktů a kvality vyvíjených produktů společnosti. Dalšími úkoly bylo automatizovat běžné činnosti ve společnosti a zajistit podporu vývojových týmů při opakujících se aktivitách. Posledním úkolem bylo definovat a nastavit procesní rámce projektového řízení ve společnosti, nástroje pro jeho podporu, kontrolní body a monitorování projektů pomocí definice vhodných výkonnostních ukazatelů. Toto vše muselo být definováno s ohledem na normy ISO 9001 a 12207 jejich dodržování je ve společnosti kontrolováno externími auditory.

Prvním úkolem byla definice procesního rámce projektového řízení a nástrojů pro jeho podporu jenž jsou popsány v kapitole 2 . Pro projektové řízení a procesní zázemí pro jeho podporu bylo nakonec zvoleno agilního přístupu a to konkrétně metodiky SCRUM (detailněji popsáno v kapitole 2 .1), jenž se orientuje na rychlé dodání funkčních částí software zákazníkovi a obecně vysokou spolupráci zákazníka a společnosti při vývoji software. Současně s volbou a transformací procesů projektového řízení probíhala volba nástrojů pro vlastní podporu Continuous Integration. Dalším úkolem byla volba vhodného integračního serveru pro podporu nově definovaných procesů. V kapitole 5 jsou nastíněny v dnešní době používané integrační servery, které připadaly v úvahu v ELVAC SOLUTIONS a jejich konkurence.

Během implementace Continuous Integration procesů v ELVAC SOLUTIONS jsem se držel zásad a doporučení specifikovaných v rámci agilního přístupu [5] a metodiky SCRUM a zásad pro úspěšnou implementaci Continuous Integration [1] [2] [4] (detailněji kapitola 4). a také specifika implementace v prostředí .NET [3] a vývoje v prostředí DevExpress (detailněji kapitola 7 .1).

Poslední kapitoly práce se pak soustředí na konkrétní implementaci Continuous Integration, dle získaných informací, ve společnosti ELVAC SOLUTIONS a návrh možností implementace obdobného systému práce pro vývoj v prostředí JAVA.

2 Procesy projektového řízení

Každý projekt je unikátní z pohledu nejen obsahu, ale i aktivit nutných k jeho dokončení. Aby ale bylo možno projekty mezi sebou porovnávat, kontrolovat jejich stav a monitorovat postupy prací, je nutno zavést a definovat fáze projektu. Projektové řízení obecně lze tedy rozdělit do čtyř základních fází:

- Zahájení (Initiation)
- Plánování (Planning)
- Realizace (Execution)
- Uzavření (Closure)

V každé fázi jsou definovány kontrolní body, které monitorují postup prací na projektu, jejich stav a sjednocují projektové řízení, bez ohledu na obsah realizace projektu.

V úvodní fázi dochází ke zhodnocení projektu a jeho očekávaných benefitů pro zákazníka i dodavatele. V této fázi dochází ke specifikaci obecných požadavků na systém, cílů projektu z pohledu zákazníka a mapování cílů na obchodní příležitosti. Ve fázi plánování projektu se již tvoří projektový tým a dochází ke specifikaci projektu v širším kontextu. V rámci realizace dochází k vlastnímu vývoji dle požadavků a v poslední fázi dochází k předání a ukončení projektu. Každou z těchto fází určuje jeden nebo více kontrolních bodů (Obrázek 1), kde dochází k vyhodnocení postupu prací na projektu, odchylek od rozpočtu, plánu, zadání a podobně. Rozhodnutí o pokračování projektu je učiněno na schůzce řídícího výboru projektu (project steering group) v rámci každého kontrolního bodu.

Při implementaci Continuous Integration a podpoře na straně projektového řízení jsem na základě zkušeností z předchozího zaměstnání definoval následující kontrolní body:

- DP1 Rozhodnutí o zahájení projektu
- DP2 Rozhodnutí o pokračování, změně nebo ukončení (přerušení) přípravných prací projektu
- DP4p Rozhodnutí o započetí prací před DP3
- DP3 Rozhodnutí o schválení projektového plánu
- DP4 Rozhodnutí o zahájení prací
- DP5 Rozhodnutí o pokračování, změně nebo ukončení projektu
- DP6 Rozhodnutí o schválení výsledku (část řešení, celý projekt, apod.)
- DP7 Rozhodnutí o změně odpovědnosti za projekt (předání, SLA apod.)

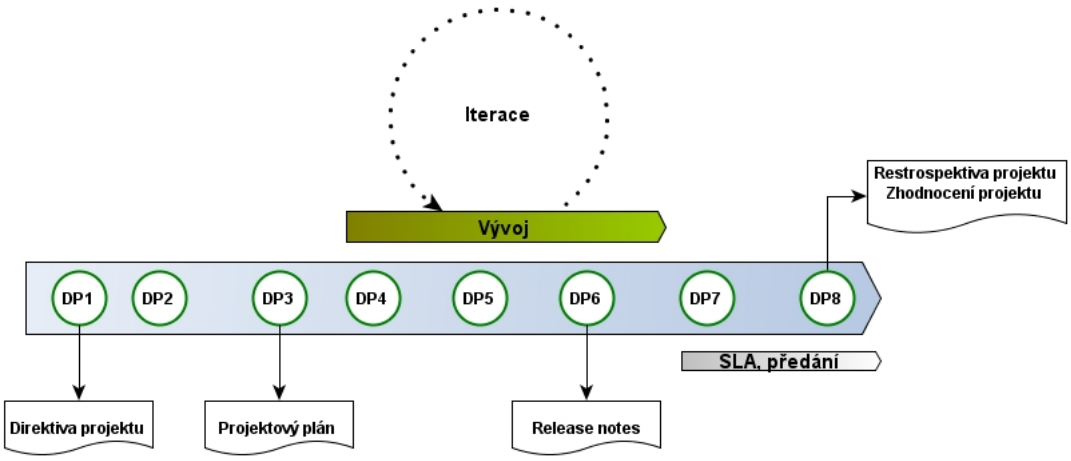


Figure 1: Kontroloní body projektového řízení

- DP8 Rozhodnutí o uzavření a ukončení projektu

V realitě se, ale setkáváme s projekty různě velkými a v případě menších projektů vytváří tento systém práce (Obrázek 1) zbytečné nároky na organizaci a řízení, které ve výsledku zvyšují náklady na projekt bez zjevného přínosu. Proto je dobré definovat typy kontrolních bodů v závislosti na předpokládané velikosti projektu. Lze například definovat, že projekty s očekávanou náročností do 100 hodin jsou považovány za malé a budou podléhat jinému způsobu řízení (Obrázek 2) a velké budou řízeny pomocí standardního nastavení kontrolních bodů (Obrázek 3).

Figure 2: Kontroloní body projektového řízení pro malé projekty

Figure 3: Kontroloní body projektového řízení pro velké projekty

2 .1 SCRUM

SCRUM je jednou z nejrozšířenějších metodik vývoje software v Agilním prostředí. Vývoj software pomocí SCRUM staví na vývoji menších částí funkčního řešení místo vývoje

celého software najednou.

Vývoj v agilním prostředí se řídí následujícími zásadami [5]

- **Individuals and interactions** over processes and tools
- **Working software**over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

Vývoj v prostředí SCRUM je tedy rozdělen do kratších iterací (sprintů) v délce maximálně několika týdnů a každá tato iterace je ukončena vydáním nové, funkční verze software (ve SCRUM terminologii - Potentionally Shippable Product). Takovýto způsob práce umožňuje vývojovým týmům soustředit se na menší části systémů (moduly a funkcionality) a zároveň rychleji získat zpětnou vazbu na již dodané části od odpovědných osob (řídící skupina projektu) a zákazníka. Každá další fáze (iterace) pak staví na již vyvinuté a schválené funkcionalitě z předchozích sprintů.

Hlavní výhodou tohoto přístupu je flexibilita a rychlá reakce na změny. K fixaci obsahu realizace dochází na krátké časové úseky a důsledkem takovéhoto způsobu práce je minimalizace zbytečné práce a realizace pouze požadovaných částí systému dle aktuálních priorit a možností zadavatele. Další nespornou výhodou pro zadavatele projektu je možnost kdykoliv projekt ukončit a mít stále funkční systém (odtud právě plyne termín potentionally shipable product), který byl vyvinut v rámci předchozích iterací. Právě zafixování obsahu každé iterace je důležitým prvkem projektového řízení pomocí SCRUM.

Řízení projektů pomocí SCRUM metodiky, nevyžaduje žádné další náklady na speciální software a v nejjednodušší formě může být projekt veden pomocí nástěnky, papíru a tužky.

2 .2 Projektové řízení v prostředí SCRUM

V rámci SCRUM jsou definovány základní role:

- Product Owner - vlastník (majitel) projektu na straně zákazníka i realizátora. Tito vedou řídící skupinu projektu, rozhodují o obsahu iterací a schvalují výsledky proběhlých iterací
- Vývojový tým - realizátoři obsahu iterace definované řídící skupinou projektu
- SCRUM Master - role, která zajišťuje kontrolu procesu vývoje a jeho vylepšování. Podává informace a stavu projektu v rámci řídící skupiny.

Za projektové řízení ve fázi realizace projektu je odpovědný SCRUM Master, jehož hlavním úkolem je kontrola dodržování procesu, implementace vylepšení procesů a komunikace s řídící skupinou. Hlavní výkonou jednotkou je vlastní projektový tým.

V této fázi je projekt (software) definován pomocí produktového zásobníku (Product Backlog), který slouží jako úložiště pro veškeré požadavky na systém. V tomto zásobníku jsou pak požadavky uvedeny ve formě uživatelských příběhů.

Jako ***správce systému****, chci mít možnost* ***zobrazit uživatele podle data posledního přihlášení****, protože to* ***usnadňuje řešení problémů s přihlášením***

Takovéto uživatelské příběhy obsahují pouze základní údaje - kdo (budoucí role v systému), jakou funkcionalitu a proč požaduje (benefit). Další informací k tomuto požadavku je priorita, která je vstupem (spolu s očekávaným benefitem) pro plánování iterace. Do produktového zásobníku mohou požadavky vkládat všichni uživatelé. Jejich prioritizace probíhá v podstatě neustále dle aktuálních potřeb zadavatele. Na jednání řídící skupiny projektu je rozhodnuto, které požadavky budou implementovány v následující iteraci a je definován zásobník iterace.

Při plánování a definici zásobníku iterace je potřeba zohlednit prioritu požadavku, očekávanou náročnost implementace a hodinovou kapacitu vývojového týmu na danou iteraci. Jakmile je obsah iterace schválen, stává se tento závazným a žádné další změny v těchto požadavcích nejsou povoleny (dochází k fixaci obsahu realizace v dané iteraci). Dalším krokem je detailnější plánování, kde jsou jednotlivé požadavky (uživatelské příběhy) analyzovány a případně s jejich zadavateli komunikovány detaily implementace. V tomto kroku dochází také k rozdělení požadavků na dílčí úlohy a tyto jsou přiřazeny jednotlivým členům vývojového týmu k realizaci a práce na iteraci jsou zahájeny. Dochází tedy ke kontrolovanému rozhodnutí o zahájení prací (kontrolní bod) a pomyslnému slibu vývojového týmu, co bude výsledkem nadcházející iterace.

V průběhu iterace se členové týmu schází k pravidelným denním poradám (ve SCRUM terminologii daily stand-up meeting). Na těchto poradách jsou sdíleny informace o postupu prací, možných problémech a prodleních. Jednání je řízeno třemi otázkami:

- Co jsem dělal včera
- Co mám v plánu dnes
- Co mi brání v pokračování (problémy)

Zodpovězení těchto otázek netrvá dlouho a zároveň umožňuje vedení projektu reagovat na vzniklou situaci již v průběhu iterace.

Na konci každé iterace je výsledek realizace prezentován zákazníkovi v rámci kontrolního bodu schválení výsledku realizace. V této fázi jsou přijímány požadavky na změnu či vylepšení a tyto jsou opět prioritizovány v produktovém zásobníku. Změnové řízení probíhá v rámci fáze realizace, kdy mimo schváleného obsahu iterace je možné doplňovat požadavky a měnit priority již definovaných úkolů. Na konci každé iterace lze učinit rozhodnutí, že další iterace již není třeba a ukončit projekt.

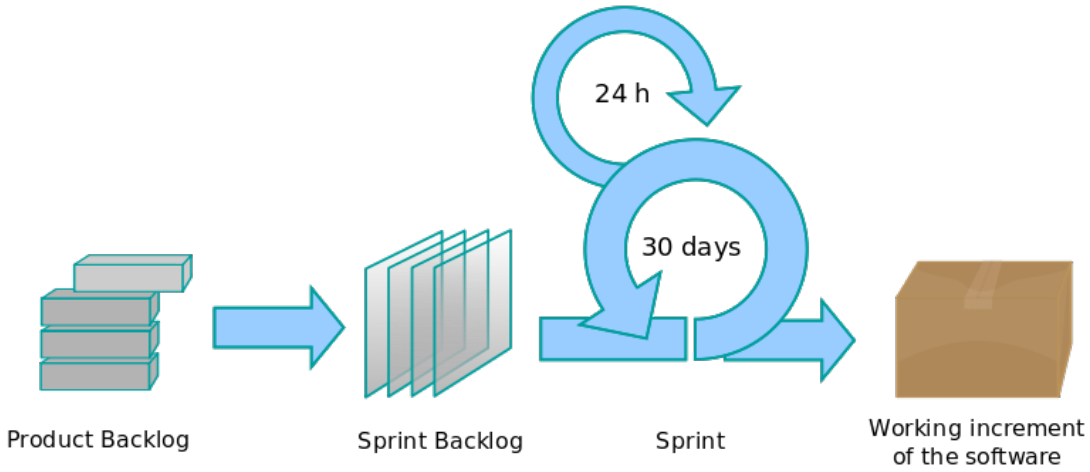


Figure 4: SCRUM proces

Každá iterace je ukončena retrospektivou, ve které projektový tým diskutuje její průběh a navrhuje vylepšení. Retrospektiva se řídí následujícími dotazy:

- Co šlo dobře?
- Co nešlo dobře?
- Co by se mělo vylepšit do příštích iterací?

Na vedoucím projektu (SCRUM Master) je následně implementace a integrace těchto vylepšení a opatření do procesu vývoje software pro následné iterace. Takto se tým organizuje a zlepšuje v průběhu realizace projektu.

2 .3 Nástroje pro podporu řízení projektů

Nástrojů pro podporu řízení projektů a vývoje software je mnoho. Volba těchto nástrojů by měla zohlednit principy vývoje software v agilním prostředí.

- Rychlou reakci na změny
- Spolupráci se zákazníkem
- Rychlý vývoj a instantní zpětnou vazbu
- Fungující software po každé iteraci

Jak již bylo naznačeno, implementace projektového řízení pomocí metodiky SCRUM není náročná na nástroje a v nejjednodušší verzi lze implementovat pomocí obyčejné nástěnky a lístků papíru. Pro rozšířenou automatizaci činností a zefektivnění vývoje bylo nutné integrovat nástroje podporující běžné činnosti v rámci vývoje software a projektového řízení. V rámci implementace Continuous Integration ve společnosti ELVAC SOLUTIONS byla zvolena kombinace nástrojů placených a open source.

2 .3.1 JIRA

JIRA [6] je placený nástroj firmy Atlassian, jenž podporuje vývoj software a projektové řízení. JIRA je modulární a lze rozšířit o další možnosti pomocí plug-in systému. V rámci JIRA lze definovat pracovní postupy, priority, komunikovat se zákazníkem, specifikovat, kontrolovat a monitorovat požadavky a jejich stav. Každý projekt v JIRA obsahuje seznam úloh (slouží tedy jako produktový zásobník) a tyto jsou pomocí definovaného pracovního postupu implementovány dle požadavků a priorit zákazníka. V ELVAC SOLUTIONS je JIRA použita nejen pro potřeby projektového řízení, ale také pro release management, zákaznickou podporu a servisní činnost a také pro monitorování projektových a servisních indikátorů (KPI - Key Performance Indicators) integrující tak veškeré procesní náležitosti projektového řízení do jednoho nástroje.

Rozšířením JIRA je plug-in TEMPO, který umožňuje kontrolu prací z pohledu odpracovaných hodin, jejich plánování, automatické pracovní postupy a schvalování výkazů práce. Právě zápis prací pomocí TEMPO je klíčový pro kontrolu výkonnosti projektových týmů z pohledu odvedené práce a nákladů na projekt.

2 .3.2 Confluence

Confluence [7] je opět nástroj společnosti Atlassian. Jde o nástroj pro kolaboraci projektových týmů. Jeho výhodou je nativní integrace se systémem JIRA, které umožňuje například vytvářet úlohu přímo z textu (například ze záznamu jednání se zákazníkem nebo prezentace výsledků po iteraci). Confluence nabízí šablony pro projektové řízení, retrospektivu projektu, sběr požadavků, zápisy z jednání a mnohé další. Umožňuje také kontrolované schvalování a vydávání dokumentů. Vše je možné také upravit pro potřeby jednotlivých projektů a také rozšířit funkcionalitu pomocí plug-in systému.

V ELVAC SOLUTIONS je Confluence využita pro projektovou dokumentaci, definici požadavků, dokumentace, release notes, příručky, zápisy z jednání, zápisy z retrospektiv projektů, kontrola a podpora procesů vývoje, spolupráci se zákazníkem a další papírové náležitosti projektového řízení.

3 Key Performance Indicators

KPI (Key Performance Indicators) neboli klíčové ukazatele výkonnosti jsou metriky, ukazatele nebo indikátory měřící výkonnost procesů a služeb v rámci organizace. Tyto metriky vyjadřují požadovanou výkonnost z pohledu kvality, efektivnosti, hospodárnosti a podobně. Volba vhodných metrik vychází ze znalosti toho, co je pro danou organizaci důležité a také jak je definován úspěch.

Příkladem takových metrik pro vývoj software, mohou být:

- Žádná chyba v produkčním release.
- 100% zákazníků je spokojeno se službami.

Takto definované metriky jsou ale dosti obecné a také nereálné.

Cílem je definovat sadu metrik tak, aby bylo možno monitorovat, kontrolovat a následně vylepšovat procesy a služby efektivně. Obzvláště nevhodné je mít mnoho metrik, kdy se čitelnost výsledků snižuje. Důležitým krokem je definovat reálné cíle a zajistit v jaké fázi procesu a jak se tyto metriky budou měřit a vyhodnocovat.

Základní principy definici KPI lze popsat následovně:

- KPI musí být orientovány na cíle, protože chceme dosáhnout určité úrovně (kvality, služeb a podobně).
- Cíle mohou být odvozeny ze standardů, interních dokumentů kvality, plánů společnosti a podobně.
- KPI musí být reálné a měřitelné.
- Cíle musí být komunikovány v rámci organizace tak, aby nedocházelo k jejich zkreslení nebo odmítání.
- Metriky musí být nezávislé (na velikosti organizace, velikosti vyvíjeného software, atd.).
- Počet metrik by neměl být příliš vysoký (obecně se doporučuje maximálně 5 metrik na proces).

S ohledem na výše uvedené jsem v rámci implementace Continuous Integration procesů ve společnosti ELVAC SOLUTIONS rozdělil metriky na dva typy. Metriky projektové - hodnotící projekty z pohledu financí, času, dodání a projektového řízení - identifikující možné problémy v budoucnosti. A metriky pro Continuous Integration, které jsou spíše metrikami kvalitativními, monitorující kvalitu procesu a kvalitu vývoje.

3 .1 Projektové indikátory

Definice metrik pro projekty vychází z hlavního cíle projektů, kterým je zisk. Dalším cílem je včasné dodání v požadované kvalitě. Třetím cílem je minimalizace režie kolem řízení projektu a další aktivity, které přímo nesouvisí s vývojem software.

Ve společnosti ELVAC SOLUTIONS jsem definoval následující KPI pro projekty:

- Odchylka (%) od plánovaného rozpočtu
- Náklady (%) na řízení projektu
- Odchylka (%) od plánovaného počtu hodin
- Počet chyb v aplikaci odhalených zákazníkem
- Počet opravných release software

Vzhledem k tomu, že uvedené indikátory metriky jsou vyhodnocovány až na konci projektu, bylo potřeba definovat ukazatele, které by pomáhaly identifikovat budoucí odchylky již v průběhu projektu. K výpočtu takových ukazatelů je potřeba znát aktuální stav prací na projektu a předpokládaný plán prací, tak jak byl projekt plánován. Takové hodnoty jsou tedy vztaženy k odhadované hodinové náročnosti prací, aktuálně odvedené práci vývojovým týmem a plánem postupu prací. Pomocí poměru jednotlivých hodnot lze kontrolovat průběh projektu a případně reagovat na problémy, které mohou nastat.

Hodnoty, na základě kterých dochází k výpočtu těchto metrik jsou následující:

- Rozpočtová cena naplánované práce (Budgeted Cost of Work Scheduled - BCWS) je definována hodnotou odhadované náročnosti realizace - například počet odhadovaných hodin násobenou interní cenou této hodiny.
- Aktuální cena vykonané práce (Actual Cost of Work Performed - ACWP) je definována pomocí hodin reálně odpracovaných na projektu a vynásobena jejich cenou.
- Rozpočtová cena vykonané práce (Budgeted Cost of Work Performed - BCWP) je tvořena cenou (počtem hodin), která je naplánována. Zde se počítá s rozdělením prací, kdy na každý (pracovní) den je určena práce a její cena. Máme-li projekt, který má rozpočet 10 000 jednotek (například korun) a jeho doba trvání je 10 dní, očekává se tedy každý den vykonaná práce za 1 000 jendotek.

Zdrojem těchto dat je systém JIRA, kde je vývojový tým povinen (v rámci definovaných procesů) ke každému úkolu uvést odpracované hodiny. Tyto hodiny jsou pak načteny do interního systému ELVAC SOLUTIONS a dochází k jejich vyhodnocení vzhledem k projektu. Vyhodnocení probíhá na základě nastavení projektu v tomto systému, kde jsou definovány požadované cíle, odhadovaná náročnost prací, časový rámec projektu, rozpočet projektu a další informace uvedené v projektovém plánu (Obrázek 5 a 6 - finanční detaily a zákazníci byli odstraněni z obrázků).

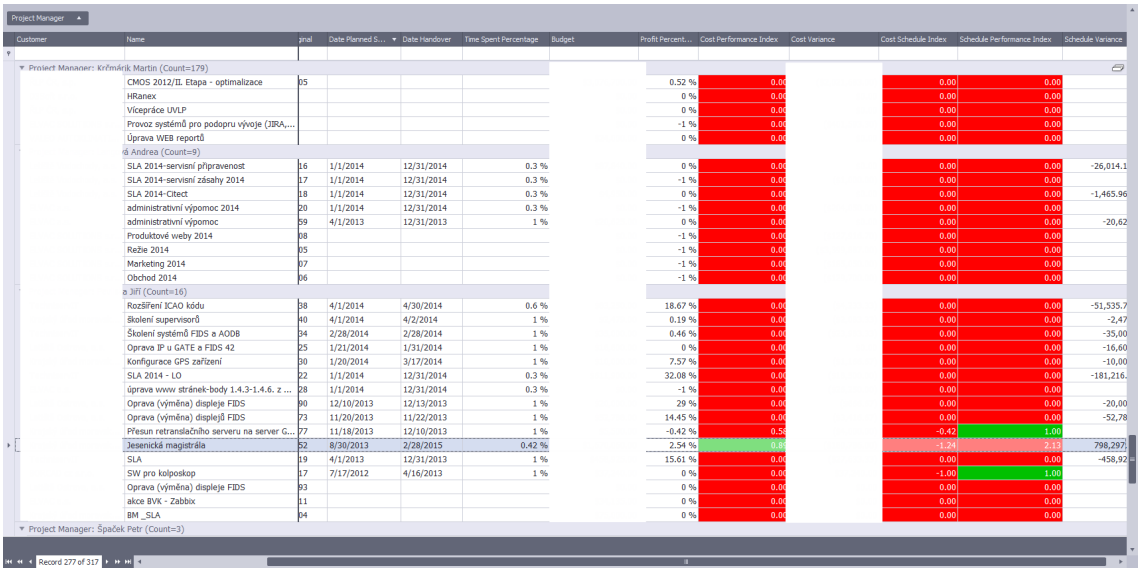


Figure 5: Visualizace projektových KPI v nástroji společnosti ELVAC SOLUTIONS

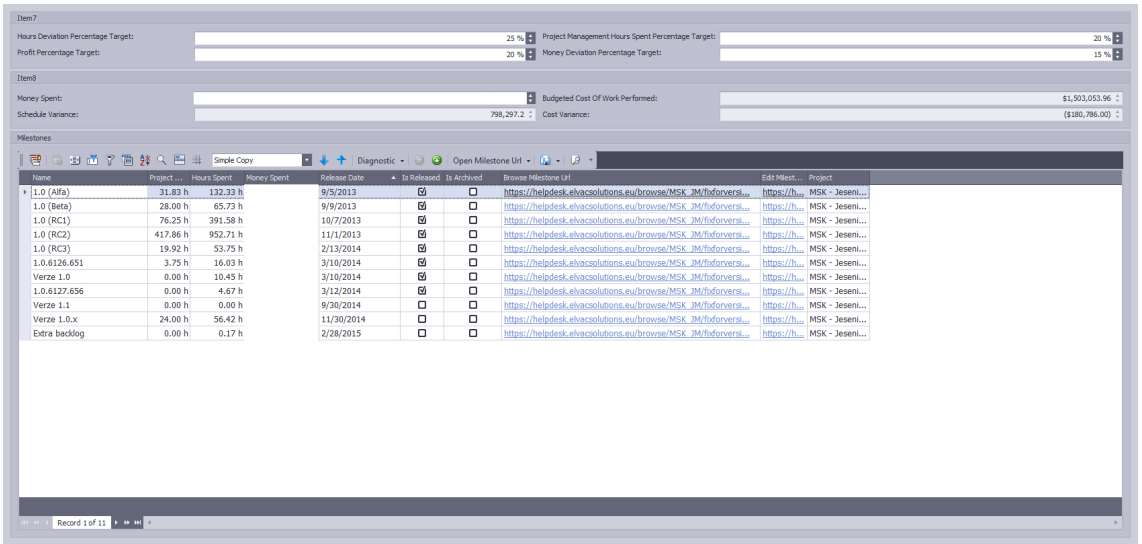


Figure 6: Detail projektu projektových KPI v nástroji společnosti ELVAC SOLUTIONS

Na základě takto získaných hodnot jsou počítány ukazatele popisující stav projektu v daném okamžiku. Tyto ukazatele jsou následující:

- Cost Performance Index (CPI) je podíl BCWP/ACWP a ukazatel efektinosti nakládání s finančními prostředky projektu. Například hodnota 1.25 ukazuje, že za každou naplánovanou jednotku (korunu) bylo z plánu 1.25 jednoky uděláno.
- Cost Variance (CV) je rozdíl BCWP a ACWP jenž ukazuje, zda je dodržován plánovaný rozpočet.
- Schedule Performance Index (SPI) je podíl BCWP/BCWS a ukazuje jak efektivně je nakládáno s časem na projektu. Například hodnota 0.67 znamená, že za každou naplánovanou hodinu práce je projektovým týmem uděláno pouze 0.67 hodiny.
- Schedule Variance (SV) je rozdíl BCWP a BCWS jenž ukazuje (ne)dodržení časového rozvrhu prací.

Obecně lze říci, že ideální hodnoty pro podílové metriky jsou kolem hodnoty 1 (ukazuje, že práce postupují, tak jak byly naplánovány) a u rozdílových metrik je to hodnota 0 (taktéž ukazuje, že projekt postupuje dle plánu - finančního nebo časového). Hodnoty pod 1 (nebo záporné hodnoty) znamenají možný problém finanční (překročení rozpočtu) nebo problém s dodáním (nedodržení termínu) a měla by se vytvořit nápravná opatření. Naopak hodnoty stabilně nad 1 (kladné) nejsou také úplně pozitivní informací. Takové hodnoty mohou poukazovat na chybu v plánu nebo opomenutí důležitého požadavku a mělo by se také přistoupit k nápravným akcím. Případně, že jsme si jisti, že jsme nic neopomenuli, takové hodnoty v tomto případě v nejmenším ukazují, že finanční a časový plán byl špatně navržen. Špatné plánování například může ovlivňovat ostatní projekty, kdy společnost žije v představě, že její pracovníci jsou vytíženi na maximum a odmítne další nabídku a to jen díky špatnému odhadu a plánu.

V rámci implementace těchto metrik se jako stěžejní ukázal nedostatek dat pro projekty. Bohužel tato data byla uložena v několika systémech a bylo nutné znovu a od začátku nadefinovat potřebné vlastnosti projektů v systému a zvolit jednotný zdroj dat (systém JIRA). V době psaní této práce je měření výkonnostních indikátorů stále ve zkušebním provozu a to právě z důvodu chybějících nebo neúplných dat, které jsou stále průběžně dodávány (viz. červené políčka v obrázku 5).

3 .2 KPI pro Continuous Integration

Také v rámci procesů a implementace Continuous Integration bylo nutné definovat kvalitativní metriky tak, aby bylo možné měřit úspěšnost těchto procesů a dopad na kvalitu vyvíjených řešení. Nástrojem pro sběr a automatické vyhodnocení kvalitativních ukazatelů byl zvolen Continuous Integration server CruiseControl.NET. Jako výkonnostní ukazatele byla zvolena následující množina vlastností zdrojového kódu:

- Pokrytí kódu unit testy
- Počet chyb v unit testech
- Index udržitelnosti kódu
- Porušení pravidel nástroje StyleCop nebo FxCop
- Rychlost sestavení aplikace

Hlavním úskalím při definici těchto ukazatelů bylo nastavení cílové hodnoty pro již fungující projekty z pohledu pokrytí kódu a dodržování pravidel StyleCop. Vzhledem k chybějícím procesům a kontrolním mechanismům ve společnosti bylo nutné zvolit přístup, kdy jsou jednotlivé hodnoty nastaveny na minimální hodnotu a kdykoliv dojde k jejímu vyhodnocení v rámci integračního serveru jako vyšší, je tato hodnota nastavena jako minimální. Například v rámci kontroly StyleCop byly nejdříve veškeré stávající zdrojové kódy daného řešení odstraněny z této kontroly. Kontrola probíhala pouze na nově vytvořených částech, kde byla naopak nastavena kontrola všech pravidel. Takto došlo k propagaci těchto pravidel mezi vývojáři a jejich konsolidaci, protože ne všechna pravidla byla vhodná. Nakonec i již vytvořené části řešení jsou postupně podstupovány kontrole a opravovány. Takto nebylo nutné opravit celé řešení z pohledu StyleCop pravidel najednou a zároveň došlo k nastavení pravidel v rámci firmy a seznámení se s nimi v rámci vývoje. Tato pravidla jsou nyní dodržována na všech nových projektech. Podobný přístup byl zvolen i v případě pokrytí kódu unit testy.

Cílové hodnoty pro počet chyb v unit testech jsou nastaveny bez tolerance a každá chyba v unit testech má za následek selhání sestavení aplikace a je nutné tuto chybu opravit.

Index udržitelnosti kódu (Maintainability index) je metrika integrovaná přímo do nástroje Visual Studio. Tato metrika je určena hodnotou z rozsahu 0-100 a reprezentuje relativní složitost údržby kódu. Hodnota 100-20 znamená dobrou udržitelnost, 19-10 střední, 9-0 značí neudržitelný kód, který by měl být přepracován. Index pro jednotlivé třídy je opět kontrolován v rámci integračního serveru a v případě zjištění nevyhovujícího indexu udržitelnosti je generováno varování protože, je obecně nevhodné nedovolit sestavení aplikace při porušení této metriky. Hlavním důvodem je vysoká náročnost na přepracování takovéhoto kódu a proto je lepší uložit toto porušení jako návrh na vylepšení do budoucnosti.

Veškeré informace - úspěšnost sestavení, počet chyb, doba trvání sestavení, počet porušení pravidel a další jsou v přehledné formě prezentovány přímo na webovém rozhraní integračního serveru.

4 Continuous Integration

Continuous integration - CI (průběžná nebo také neustálá integrace) je proces ve vývoji softwarových systémů, kdy každý člen týmu integruje svoji část práce do systému co nejčastěji. Tento způsob vede k mnoha integracím a každá tato je ověřena automatickým sestavením software, testy, kontrolou kvality a automatickým nasazením software do testovacího (nebo produkčního) prostředí. Neustálou integrací zdrojových kódů částí řešení (nazývaných check-in nebo commit), lze téměř ihned identifikovat nefunkční či problematické části (a jejich autora) a ihned reagovat jejich odstraněním.

Protože výsledek takovéto integrace je znám velmi rychle v porovnání se standardní cestou vývoje, dochází k minimalizaci časové ztráty, ke které dochází v případech dohledávání původu nastalých problémů po delším časovém úseku (například až před vlastním release aplikace) a zvyšuje tak výkonnost vývojových týmů.

Vlastní implementace podpory Continuous Integration není příliš nákladná. Naopak nevyužívání procesů Continuous Integration vývojovými týmy může vést k prodlením ve fázích projektů, kdy je software integrován do celku (například pro účely prezentace software zákazníkovi) a kdy identifikace chyby a její následná oprava je mnohem nákladnější, a časově náročnější, než při včasném odhalení [2]. Navíc, i když se podaří chybu odhalit v rozumném čase, tato může do systému jako celku zanést další chyby a nastávají tak další prodlevy.

4 .1 Pravidla Continuous Integration

Continuous Integration není proces v pravém slova smyslu, ale představuje spíše sadu principů a doporučení, které při správné implementaci přinášejí výhody v ušetřeném čase a tedy zvýšené produktivitě, kvalitě dodávaného software a rychlosti s jakou lze tento software doručit. Aby tento způsob vývoje byl pro společnost či vývojový tým platný a přinášel očekávané výhody, jsou definována některá základní pravidla [1] [2] [4].

Udržovat jediné úložiště zdrojových kódů

Softwarové systémy obsahují spoustu souborů, které je potřeba sestavit dohromady k tomu aby vytvořili výsledný produkt. Sledování všech těchto souborů je náročný úkol, zvláště pokud se na vývoji podílí více lidí. Proto je pro potřeby Continuous Integration nutné využívat nástroje správy zdrojových kódu (Source Configuration Management - SCM) jako je SVN, CVS nebo git.

Toto úložiště je pak využíváno jako zdroj pro automatizované sestavení a další aktivity Continuous Integration. Proto je nezbytně důležité, aby na tomto místě bylo vše (testovací skripty, knihovny, databázové schémata, instalační skripty apod.) co je potřeba pro sestavení a nasazení software.

Nástroje pro správu zdrojových kódů (SCM) umožňují vývojovým týmům vytvářet větve (branch) software pro potřeby oprav chyb či větších úprav kódu. Sestavení aplikace pak musí probíhat nad hlavní větví řešení (nazývané trunk). Tato by měla být v každém okamžiku sestavitelná do fungujícího řešení, která neobsahuje chyby či je v určité nekompletní a neověřené fázi úprav, tak aby si kdokoliv mohl tuto větev stáhnout a pokračovat ve vývoji [2].

Automatické sestavení

Sestavení řešení do funkčního celku obvykle obsahuje několik částí - nastavení aplikace, přesun souborů, nastavení verze produktů, sestavení, vytvoření databázových schémat, import základních dat a nastavení, vytvoření release notes a další. Tyto úkony jsou časově náročné v případě manuálního vykonávání a to je pro potřeby opakovatelného a jednotného procesu sestavení aplikace ve funkční celek (a dalších náležitostí) nevhodné. Proto je jedním ze základů funkční implementace Continuous Integration automatizace běžných úkonů, tak aby bylo možno je vykonat spuštěním jediného příkazu bez manuální interakce. Zde platí stejné pravidlo jako v případě úložiště zdrojových kódů a to, že každý by měl mít možnost vykonat tento příkaz lokálně na svém počítači [2] (například pro potřeby ověření výsledku ještě před integrací na serveru).

V tomto případě není vhodné používat nástrojů, které v sobě obsahují vývojová prostředí, protože většinou nejsou instalovány na cílovém počítači, ale použít obecného řešení jako je Ant, make, NAnt, MSBuild, rake a podobných. Tyto nástroje umožňují uživateli zvolit si, jakým způsobem bude aplikace sestavena - jestli proběhnou testy, jestli se bude aktualizovat databázové schéma, jestli se mají sestavit i zavislé projekty a podobně. Lze také parametrizovat sestavení aplikace (build) a tak vytvořit různé sady sestavení pro různé potřeby v závislosti na fázi projektu. Tohoto efektu se také využívá při následné definici různých úloh na integračních serverech v rámci Continuous Integration.

Automatické testy

V základním pojetí build aplikace znamená kompilaci zdrojových kódů, propojení závislostí a dalších úkonů nutných k tomu, aby se aplikace dala sestavit a následně spustit. Na druhou stranu, i když lze program spustit, neznamená to, že správně funguje. Proto je nutné každé sestavení aplikace zároveň i otestovat [2]. Tento přístup byl zaveden s příchodem technik vývoje software jako je eXtreme Programming (XP) a Test Driven Development (TDD), které jsou založeny na tom, že dříve než je napsán jediný řádek zdrojového kódu, jsou napsány testy testující tento kód a jeho budoucí funkcionalitu. Při sestavení, se pomocí těchto předem definovaných testů, testuje nově napsaný kód a build aplikace se nezdaří, pokud neprojde testy.

Toto je pochopitelně postačující pro jednotlivé funkcionality, ale pro otestování řešení jako celku je potřeba nejen testů jednotlivých funkcionalit (unit testů), protože funkčnost celku obvykle není zaručena funkčností jednotlivých jeho částí. Proto je nutné, na úrovni

aplikace, definovat integrační testy (popřípadě další typy jako jsou výkonnostní testy), které prověřují aplikaci jako celek. Pro potřeby Continuous Integration je opět nezbytné, aby takovéto testy bylo možno spouštět na integračním serveru jediným příkazem a také aby bylo možné definovat skupiny testů, které jsou spuštěny pro potřeby jednotlivých fází vývoje.

Pro automatické testování v průběhu sestavování aplikace je nejvhodnějším řešením rodina testovacích nástrojů jako je JUnit, NUnit, Microsofot Visual Studio UnitTesting Framework a podobné. Pro definici integračních a end-to-end testů lze využít open source nástrojů jako je Selenium, FITnesse, EasyTest, WatiN (a jeho implementace v jazyce Java - WatiJ nebo Ruby - Watir) nebo komerčních produktů jako je Microsoft Test Manager, HP TestRunner a další.

Častý commit změn

Continuous Integration je svým způsobem komunikací vývojářů, jenž umožňuje propagovat mezi členy týmu odvedenou práci a změny, které byly provedeny v řešení [2]. Pokud je toto prováděno často a pravidelně, chyby a problémy pramenící ze zásahu dvou vývojářů do jedné části řešení jsou odhaleny rychle a následná oprava je neporovnatelně jednoduší než v případě, kdy takovéto konflikty zůstanou skryty několik dnů. Důvodem je hlavně minimalizace počtu změn, které byly provedeny v kódu v daném období a také commit provedený každých pár hodin obsahuje méně změn v méně souborech, než commit provedený jednou za den.

Jako podpora těchto častých commmitů se v rámci mé práce v ELVAC SOLUTIONS osvědčilo umožnit vývojářům provést sestavení na svém počítači stejnou cestou, jako bude proveden na integračním serveru - tedy voláním stejné sekvence úloh sestavovacího skriptu včetně testů a lokálního nasazení software. Tento přístup ještě více minimalizuje prodlevy ve zpětné vazbě.

Další výhodou častých commitů je rozdělení práce na menší úseky, což usnadňuje vhled do postupů prací při vývoji a pomáhá při řízení projektu v odhadu možných problémů s odevzdáním či zdroji.

Každý commit musí být sestaven na integračním serveru

Ačkoliv, jak bylo zmíněno v předchozím bodě, je možnost lokálního sestavení aplikace vhodnou praktikou při implementaci Continuous Integration, hlavním a určujícím faktorem v rozhodování o tom, zda build byl či nebyl úspěšný (a tedy práce vývojáře schválena) je na integračním serveru. Toto pramení z rozdílů mezi osobními počítači jednotlivých vývojářů, ale například také z nedodržování pravidel, kdy před lokálním sestavením si vývojář musí nejprve stáhnout z repositáře aktuální verzi vyvíjeného software, vyřešit případné konflikty se změnami ostatních a pak aplikaci sestavit a otestovat. Z těchto důvodů, je nutné každý commit ověřit na nezávislém stroji a teprve pokud na

tomto stroji je sestavení úspěšné (včetně všech náležitostí jako jsou testy, nové schéma databáze apod.) je commit (odvedená práce) považován za úspěšný a hotový.

V případě Continuous Integration se používá dvou přístupů, a to je manuální sestavení a automatické sestavení pomocí integračního serveru. V případě manuálního sestavení jde o podobný postup, jako v případě lokálního sestavení vývojáři před commitem změn. Odpovědný vývojář se připojí k integračnímu stroji, stáhne si aktuální verzi zdrojových kódů řešení z repositáře, vyřeší případné konflikty a spustí integrační build. Po té sleduje sestavení aplikace a pokud vše proběhne v pořádku, může považovat svoje změny za schválené. Tento přístup ale není moc praktický a je nutná investice času jednoho člena vývojového týmu.

V případě serveru pro Continuous Integration jde o pravidelnou a automatickou kontrolu hlavní větve (trunk) řešení v repositáři. V případě změny si integrační server automaticky stáhne nejnovější verzi a spustí integrační build využívající předem definovaných skriptů. Autor změny je následně vyrozuměn o výsledku této integrace. Klíčovou částí tohoto přístupu je, že v případě, kdy sestavení aplikace selže musí být hlavní vývojová větev (trunk) ihned opravena. Takto zaručujeme, že se vyvíjí vždy na stabilním základě (sestavitelný, testy ověřený zdrojový kód), ale také zabraňujeme hromadění chyb v hlavní větvi a tedy blokování práce ostatních vývojářů, kteří nemohou začít pracovat nad nefunkčním řešením.

Jako podpora tohoto přístupu se využívá funkcionalit ”pending head” [2] či ”delayed commit” [2], kdy změny jsou reálně vloženy do hlavní větve repositáře pouze pokud projdou sestavením na integračním serveru. Jsou tedy po dobu integrace drženy v odděleném repositáři, který není veřejně přístupný. Tohoto lze momentálně využít u placených nástrojů JetBrains TeamCity nebo Atlassian Bamboo. Nespornou výhodou tohoto přístupu je, že hlavní větev řešení bude vždy stabilní a sestavitelná ve fungující řešení.

Jako alternativu k placeným nástrojům, lze využít například takzvaných ”pre-commit hooks” na straně repositáře, kdy před vlastním commitem je aplikace sestavena. Toto bohužel prodlužuje vlastní commit a v praxi se neukázalo jako vhodné řešení při déle trvajících buildech, či u složitějších aplikací s mnoha závislostmi. Také lze využít oddělených repositářů pro integraci (neveřejný) a jako trunk (veřejný). Konfigurace tohoto přístupu je však složitější a vyžaduje spolupráci vývojářů - například commit nesmí jít do trunku, ale neveřejného repositáře.

Optimalizace sestavení

Zásadní myšlenkou Continuous Integration je poskytnutí rychlé zpětné vazby [1]. S ohledem na výše zmíněná doporučení, kdy například práce (commit) není hotova dokud neprojde sestavením na integračním stroji, je velmi nežádoucí čekat na build aplikace více než několik minut. Toto doporučení je v některých případech velice těžké dodržet, protože některé, zvláště testovací, úlohy zabírají spoustu času. V průběhu implementace

Continuous Integration procesů v ELVAC SOLUTIONS jsem věnoval značný čas těmto optimalizacím a zrychlováním sestavení řešení pomocí distribuovaného sestavení, paralelních build procesů a optimalizací závislostí projektů. Výsledkem je průměrná doba na build aplikace, včetně nasazení, kolem 8 minut a nejdelší sestavení aplikace trvá kolem 19 minut.

Hlavním problémem při zrychlování buildu aplikace jsou integrační (end-to-end) testy, z nichž některé jsou časově velmi náročné (v některých případech kompletní funkcionální otestování aplikace zabere i několik hodin). V ELVAC SOLUTIONS jsem proto implementoval několik front pro sestavení aplikace [2]. V první je aplikace sestavena, jsou provedeny unit testy a je nasazena do testovacího prostředí. Pokud je tento build úspěšný jsou následně na nezávislém stroji spuštěny další integrační a funkcionální testy, které slouží k dalšímu ověření integrace. Z časových důvodů se pro ověření a schválení commitu používá právě sestavení z první fronty. Tuto praxi je také možné nahradit pomocí tzv. nočních buildů (nightly build), kdy se aplikace ověří jako celek a ráno jsou vývojářům prezentovány výsledky.

Zpřístupnit aplikaci ostatním

Dalším úskalím vývoje software je identifikace, zda opravdu vytváříme "správné" řešení. Ve většině případů je téměř nemožné mít předem přesné specifikace požadavků na aplikaci. Pro většinu je naopak mnohem snadnější vidět aplikaci za běhu a na tomto základě rozhodovat co a jak by mělo vypadat. Proto je žádoucí prezentovat aplikaci co možná nejčastěji.

Aby toto bylo umožněno je velmi důležité zpřístupnit vyvíjený software všem zainteresovaným stranám ať už pro testování, demonstraci nových funkcionalit nebo pro kontrolu co se za danou fázi vývoje změnilo. Proto je důležité definovat úložiště ze kterého si interní i externí uživatelé mohou nové verze software stáhnout a vyzkoušet a toto důkladně komunikovat interně i směrem k zákazníkům. Rozhodnutí, které verze aplikace prezentovat by se mělo řídit integračním sestavením z druhé fronty (tedy ve chvíli, kdy proběhnou integrační, end-to-end a další testy), a kdy se předpokládá, že sestavení je úspěšné a dostatečně stabilní, aby mohlo být prezentováno.

Každý musí vidět co se děje

Použití Continuous Integration jako prostředku pro rozšíření komunikace mezi vývojáři vyžaduje zajištění prezentace stavu systému a změn v něm provedených všem zúčastněným. Nejdůležitější informací je stav hlavního buildu řešení. Některé nástroje nabízejí webová rozhraní nebo aplikace, které indikují stav sestavení řešení. Ve velkých firmách mohou najít uplatnění obrazovky zobrazující stavy sestavení a front.

Automatizace nasazení

Jak již bylo zmíněno, pro potřeby Continuous Integration je vhodné využívat několik prostředí pro sestavení, testy, prezentace a podobně. Vzhledem k tomu, že dochází k tomuto nasazení aplikace i několikrát denně je logickým krokem automatizace těchto úkonů [1]. Automatizace těchto úkonů je možná pomocí různých skriptovacích jazyků - PowerShell, bash, cmd, ruby a další. Výhodou takto implementovaného automatického nasazení je možnost využít těchto skriptů i při nasazení produkční verze k zákazníkovi. I když se takové nasazení neprovádí každý den, je tento způsob úsporou času, protože skripty jsou již vytvořeny, dochází ke zrychlení nasazení a omezují se chyby způsobené lidským faktorem.

5 Servery pro Continuous Integration

Continuous Integration jako způsob vývoje softwarových systémů v sobě zahrnuje využití kombinace nástrojů, tak aby bylo dosaženo požadovaného efektu. Mezi základní patří správa zdrojových kódů (source control managment - SCM) pomocí SVN, git, CSV, TFS a dalších. Dále nástroj pro sestavení aplikace jako je Ant, NAnt, MSBuild, Maven nebo rake. Využití vhodného testovacího nástroje pro automatizované unit a end-to-end testy jako jsou NUnit, JUnit, Selenium, EasyTest, FITnesse a podobné. Vhodného skriptovacího jazyka pro automatizaci nasazení software jako je PowerShell nebo bash a další podpůrné nástroje - například pro automatické generování a publikaci uživatelské dokumentace, vytváření release notes aplikace a podobné.

Výše zmíněné nástroje a skripty jsou pak integrovány do celku v rámci běhu integračního serveru zajišťujícího jádro podpory procesů Continuous Integration. Tento server funguje na principu kontroly definované větve řešení v repositáři (většinou trunk) v pravidelných intervalech. V případě, že v definované větvi došlo ke změně, je tato stažena na build server a sestavena pomocí volání jednotlivých akcí build skriptů (Ant, NAnt, MSBuild apod.). Tyto servery jsou koncipovány modulárně a umožňují rozšíření své funkcionality pomocí nejrůznějších doplňků (plug-in), které jsou vyvíjeny komunitami sdružujících se kolem těchto nástrojů nebo je lze, v případě open source řešení, vytvářet i nezávisle podle potřeb vývojových týmů.

Většina nástrojů (placených nebo neplacených) poskytuje přibližně stejnou sadu funkcí jako je integrace s build nástroji, bezpečnostní profily (ve více či méně omezené formě), podpora testovacích nástrojů, podpora pro nástroje statické nebo dynamické analýzy kódu a většina již nabízí možnost distribuovaných sestavení. Proto je hlavním kritériem pro rozhodnutí, který nástroj zvolit, platforma, ve které daný tým či společnost vyvíjí své aplikace. Výhodou placených nástrojů je jejich snadná konfigurace, přehledné uživatelské rozhraní a konfigurace úloh s množstvím různých nastavení. Na druhou stranu, toto je možné doprogramovat do serverů, které jsou zdarma, ve chvíli, kdy je tato funkcionalita vyžadována.

5 .1 CruiseControl

CruiseControl (CC) a CruiseControl.NET (CCNET) jsou open source servery navržené pro řešení v jazyce Java respektive .NET firmou ThoughtWorks. Veškerá konfigurace serveru a úloh je řešena v XML souborech. Obsahuje webovou komponentu umožňující sledovat průběh integrace. Pro uživatele je navíc možnost nainstalovat program CCTray, který umožňuje konfiguraci úloh, které chce daný uživatel sledovat a přehlednou notifikaci o jejich stavu v nástrojové liště operačního systému. Na druhou stranu zpracování webového rozhraní není na úrovni ostatních nástrojů. Nutnost konfigurace úloh pomocí zápisu v XML souboru a obecně větší složitost konfigurace má za následek delší čas nutný k nastavení a implementaci Continuous Integration pomocí serverů CruiseControl.

Velmi kladně ale hodnotím, že tento nástroj nemá žádná omezení co do počtu front, build úloh a uživatelů. CruiseControl lze nainstalovat jako službu, nativně podporuje statickou analýzu kódu, autentizaci pomocí LDAP, bezpečnostní profily ve kterých lze definovat uživatele či skupiny a jejich oprávnění k akcím či které úlohy budou zobrazeny a spoustu funkcionalit, které se hodí i pro vývoj firemních aplikací ve středních či větších firmách. Nakonec konfigurace pomocí XML, ačkoliv se nemusí zdát jako uživatelsky přívětivá umožňuje jednoduchou automatizaci a vytvoření požadovaných úloh pomocí skriptu ihned na začátku projektu či v případě většího vývoje ve větvi řešení. Velkou výhodou je podpora distribuovaných front, kdy lze připravit několik instancí CruiseControl a na nich provádět integrační úlohy paralelně. Další výhodou je velmi široká podpora SCM, široká podpora skriptovacích jazyků a rozsáhlá knihovna XSLT transformací pro prezentaci výsledků sestavení ve webovém rozhraní.

5 .2 Apache Continuum

Apache Continuum je další z open source serverů, který je primárně navržen pro řešení v jazyce Java. Co se funkcionality týče je velmi podobná serveru CruiseControl - bezpečnostní profily, release management, integrace s build nástroji jako je Maven, Ant, rake a široká podpora SCM nástrojů. Pro instalaci a úvodní nastavení je opět nutné provést konfiguraci pomocí XML souboru a je nutno mít nainstalované Java Runtime Environment (JRE). Webové rozhraní je mnohem přívětivější a umožňuje vytváření a konfiguraci jednotlivých build úloh. Jeho použití není tak rozšířené jako v případě CruiseControl nebo serveru Jenkins (5 .3).

5 .3 Jenkins

Jenkins (dříve Hudson) je další open source integrační server pro řešení vytvořené v jazyce Java. Co do funkcionality se neliší od Apache Continuum nebo CruiseControl. Ale konfigurace a nastavení, kdy vše je možné nastavit přes webové rozhraní, z něj dělají nejpoužívanější open source integrační server pro podporu vývoje alespoň v oblasti Java řešení. Jeho architektura je založena na velkém množství různých rozšíření, což umožňuje i jeho použití pro řešení v .NET nebo ruby. Další velkou výhodou je vizualizace změn ve zdrojových kódech na stránce úlohy a plugin management zobrazující nainstalované a aktivní rozšíření systému nebo úlohy.

Ačkoliv konfigurace pomocí webového rozhraní je velmi pohodlná, Jenkins umožňuje provádět úpravy i na úrovni konfiguračních souborů a tedy podporuje automatizaci zmíněnou u serveru CruiseControl.

5 .4 Bamboo

Atlassian Bamboo je komerční nástroj a jeho největší výhodou je integrace s ostatními nástroji Atlassian jako je JIRA (2 .3.1) a Confluence (2 .3.2), které jsou široce využívány pro potřeby řízení projektů. Kromě podpory Continuous Integration se tento nástroj

soustředí také na podporu Continuous Delivery, kde lze definovat prostředí (testovací, produkční, migrační a další), na které má být řešení nasazeno a to včetně nastavení práv na konkrétní prostředí pro konkrétního uživatele (například vedoucí testovacího týmu může provést nasazení do testovacího prostředí, ale ne do produkčního). Bamboo také nativně podporuje tzv. "delayed commit" (4 .1), kdy se do hlavní větve repositáře dostane pouze ten kód, který prošel všemi testy a chrání tak hlavní větev před chybami. Další nadstandardní funkcionalitou jsou automatizované merge (spojení) větví SCM do hlavní větve v případě úspěšného sestavení. Vše je prezentováno ve velmi příjemném webovém rozhraní, které umožňuje konfiguraci veškerých vlastností sestavení.

5 .5 TeamCity

JetBrains TeamCity je další komerční server pro Continuous Integration. Tento server podporuje veškeré vývojové platformy jako je Java, .NET, Ruby, C++, Python a to včetně mobilních zařízení. Široká podpora SCM nástrojů a příjemné uživatelské rozhraní je u placených nástrojů samozřejmé. TeamCity, stejně jako Bamboo, podporuje "delayed commit" (4 .1) takže k opravdovému commitu kódu řešení dojde až po integraci. Velmi rozsáhlé jsou i možnosti reportů výsledků buildů, výsledků testů, vizualizace změn ve zdrojových kódech a to i včetně historických dat a tvorby přehledů. Další výhodou je komunitní modul dovolující komunikaci a spolupráci vývojářů při řešení problémů hlavně v distribuovaných týmech.

5 .6 Team Foundation Server

Microsoft Team Foundation Server (TFS) a jeho rozšíření Team Foundation Build je produkt sloužící pro podporu Continuous Integration pro projekty vyvíjené pomocí Visual Studio. Velkou výhodou je integrace se stávajícími produkty Microsoft Team Foundation Suite jako jsou SCM, Test Manager, Test Lab Manager a další včetně Visual Studio. Vše je tedy možné definovat již při vývoji a to od testů, analýzy kvality kódu až po nasazení aplikace na testovací prostředí. Jeho velkou nevýhodou je podpora pouze nástroje MSBuild, což jej činí použitelným pouze pro vývoj ve Visual Studio.

5 .7 Cena nástrojů podporujících Continuous Integration

V případě Continuous Integration lze ke všem komerčním nástrojům najít ekvivalent v open source komunitě mající podobnou funkcionalitu. Nejoblíbenější open source nástroje jako je CruiseControl(.NET) a Jenkins mají stabilní komunitu, která software udržuje a neustále rozvíjí. Jejich výhodou je také otevřenost kódu a již zmíněná modularita umožňující doplnit požadovanou funkcionalitu vlastním vývojem.

V případě využití open source nástrojů (a to včetně správy zdrojových kódů, skriptovacích jazyků apod.) lze náklady na implementaci CI minimalizovat v podstatě pouze na cenu hardware pro různá virtuální testovací prostředí.

Při implementaci Continuous Integration v ELVAC SOLUTIONS bylo využito právě open source nástrojů (v rámci minimalizace nákladů). Nejefektivnější se (i vzhledem k vývoji v nad platformou .NET) ukázala kombinace CruiseControl.NET, NAnt, VisualSVN, PowerShell a MSBuild. V praxi při každodenním využití těchto nástrojů nebylo zatím zjištěno žádné omezení, které by muselo být řešeno placeným nástrojem třetích stran.

6 Implementace Continuous Integration v malé společnosti

Většina společností při prvotních úvahách o implementaci Continuous Integration pro projekt nebo pro celou společnost přemýšlí, zda se jim investice vyplatí a zda je vhodná pro typ činností které vykonávají. Ze zkušeností při implementaci těchto procesů ve společnosti ELVAC SOLUTIONS jsem sestavil otázky, které mohou pomoci v rozhodnutí o implementaci Continuous Integration.

- Pracujete bez systému řízení zdrojových kódů? Nelze se vrátit ke změnám nebo nelze zjistit kdo a jaké změny provedl v řešení?
- Nasazujete aplikaci manuálně do testovacího nebo dokonce do produkčního prostředí?
- Vydáváte k aplikaci spoustu opravných aktualizací nebo objevujete problémy před nasazením?
- Provádíte veškerou konfiguraci aplikace manuálně (navíc v produkčním prostředí)?
- Slýcháte často větu "'U mě na počítači to ale funguje"'?
- Pracuje na projektu více vývojářů nebo jsou týmy umístěny v různých lokalitách?
- Bude vyvíjené řešení integrováno s ostatními systémy?
- Vyvíjíte produkt?

Obecně lze říci, že pokud na jednu otázku odpovíte kladně, investice do implementace Continuous Integration se vyplatí. A jak již bylo zmíněno investice to není příliš velká s porovnání s problémy, které nastávají v případě chybějícího nastavení procesů Continuous Integration.

Hlavní výhody Continuous Integration pro společnosti zabývající se vývojem software jsou:

- Pomáhá zvyšovat kvalitu a redukovat rizika při vývoji a před odevzdáním nebo nasazením aplikace.
- Tyto procesy jsou znovupoužitelné i pro ostatní projekty a jedna implementace může být použita pro další a další projekty společnosti. Investici tedy stačí provést pouze jednou, ale lze používat neustále.
- Definicí Continuous Integration lze odhalit problémy v procesu vývoje software nebo tento chybějící proces definovat.
- Automatizace procesů při vývoji software přináší značnou úsporu času (i několik dní) a zajišťuje jednoduchou kontrolu a opakovatelnost.
- Umožňuje kontrolu a monitorování prací na projektu, kolaboraci se zákazníkem a celkovou lepší viditelnost aktivit v rámci vývoje software.

7 Implementace Continuous Integration v prostředí .NET

7 .1 Platforma DevExpress

Vývoj software ve společnosti ELVAC SOLUTIONS je založen na framerworku DevExpress a platformě .NET.

DevExpress je určena pro rychlý vývoj software na platformě .NET. Aplikace jsou založeny na modelování procesů a aktivit zákazníka a jejich transformací na objekty uložené v databázi. Aplikace vyvinuté nad touto platformou se skládají z několika hlavních částí (Obrázek 7).

- Klientská aplikace v podobě desktop aplikace nebo webové rozhraní.
- Windows služba zajišťující vykonávání pravidelných úloh nad daty, reportování, definice a kontrola pracovních postupů a další úkony.
- Databáze jenž ukládá business objekty, jejich vazby a data a slouží jako zdroj dat pro následné analýzy a reporty.

Databázový server je základním prvkem aplikací vyvinutých v prostředí DevExpress a je zde uložen business model řešení a vlastní data. Klientské aplikace (desktop nebo web) oproti této databázi ověřují svoji verzi (aktuálnost business modelu) a umožňují práci s daty aplikace. Windows služba slouží k provádění pravidelných výpočtů nad daty a další pravidelné úlohy.

Z pohledu Continuous Integration je tedy nutné sestavit desktop aplikaci a publikovat tuto na sdílené úložiště, tak aby se klienti v případě změny verze mohli aplikaci aktualizovat. V případě web aplikace je nutno publikovat soubory na server IIS (Internet Information Services). Následně je nutná konfigurace klientů (spojení s databází, umístění souborů aktualizace a nastavení web aplikace). Na úrovni databáze musí proběhnout její aktualizace v případě změny business modelu či změna verze aplikace, tak aby klientské aplikace měly informaci o nové verzi. Při vydání nové verze musí také proběhnout aktualizace Windows služby a její konfigurace (spojení k databázi). Veškeré tyto aktivity byly zohledněny při implementaci Continuous Integration procesů.

Z pohledu Continuous Integration vypadá jako nejdůležitější fáze projektu jeho realizace, ale nastavení procesů podporující Continuous Integration se dotkne všech fází ve vývoji software - od zahájení projektu až po jeho ukončení předáním díla zákazníkovi a následnou případnou podporou provozu řešení. Každý projekt je unikátní a má svá specifika, ale právě vhodným nastavením procesů projektového řízení a vývoje software je dosaženo sjednocení na úrovni kontroly, monitorování a řízení. Je tedy možné projekty porovnávat z pohledu výkonu, kvality a sledování dopadů vylepšení procesů (implementovaných v rámci SCRUM metodiky vývoje software).

Nastavení procesů a podpory Continuous Integration ve společnosti ELVAC SOLUTIONS

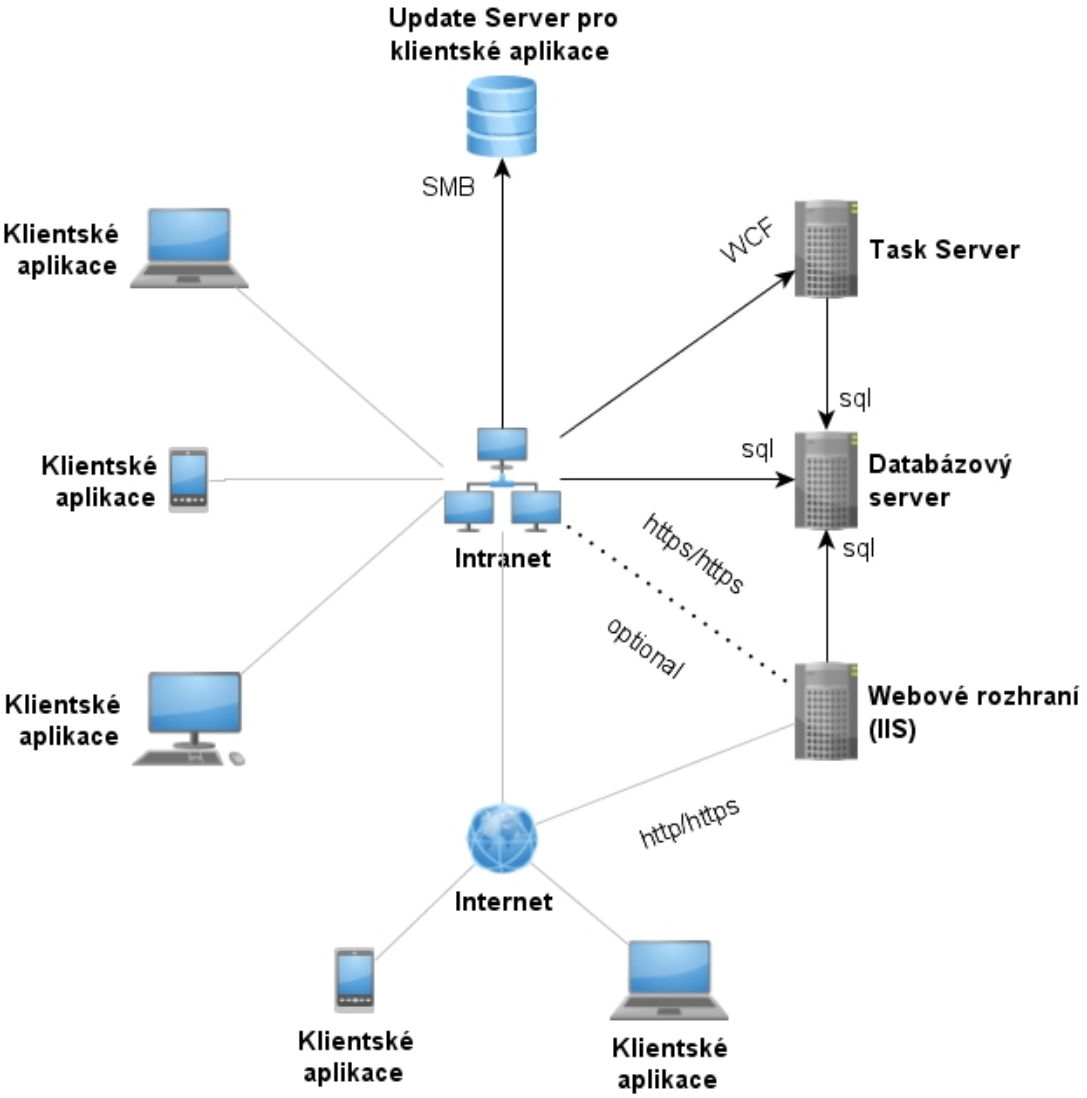


Figure 7: Architektura systémů DevExpress

probíhalo na několika úrovních a obecně je lze rozdělit právě podle fází vývoje software. V každé fázi bylo nutné identifikovat opakované aktivity, nutné dokumenty z pohledu interních procesů a procesů normy ISO 12207, navrhnout procesní rámec a identifikovat a nastavit nástroje pro jejich podporu [3].

7 .2 Zahájení projektu

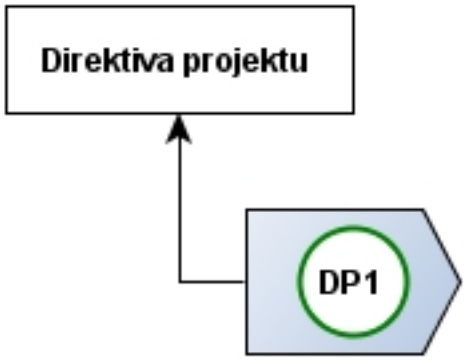


Figure 8: Direktiva projektu jako výstup zahájení projektu

Každý projekt je formou rizika a to jak pro zadavatele tak pro realizátora. Zadavatel investuje do projektu finance, čas a mnohdy i procesní a výrobní postupy a zkušenosti a očekává jisté přínosy - zrychlení výroby, náskok na trhu, podporu interních procesů, konkurenční výhodu a podobné. Realizátor se svojí účastí v projektu zavazuje právě tyto přínosy naplnit. Investicí realizátora je tedy krom svých zkušeností, pracovníků, času a financí hlavně pověst, důvěryhodnost a obraz společnosti na trhu. Pochopitelně úspěšnou implementací a naplnění očekávání zákazníka má pro společnost pozitivní dopad v podobě obrazu na trhu, možné následující spolupráce, reference a podobné. Ve skutečnosti, ale tyto očekávané přínosy nejsou viditelné ihned a je nutné hned ze začátku projektu začít dialog s předpokládaným zákazníkem. Mnoho společností právě tento krok podceňuje, jelikož jde o činnost, která je čistou investicí společnosti a nezaručuje získání projektu. Právě ale definicí těchto přínosů a jejich priority pro zadavatele lze zhodnotit projekt dříve než se přistoupí k dalším investicím v následujících fázích a například lze rozhodnout, že pro společnost není výhodné projekt realizovat či že nemá dostatečné zdroje. Tímto nejen, že se předejde problémům způsobeným bezhlavou realizací jakéhokoliv projektu, ale společnost získává v očích objednatele obraz korektního partnera i když se rozhodne projekt odmítnout.

Ve fázi zahájení projektu tedy dochází právě k zhodnocení projektu a to hlavně z pohledu očekávaného přínosu jeho implementace pro zadavatele a řešitele. V rámci tohoto hodnocení jsou hlavní faktory finance, obsah implementace, schopnosti a zkušenosti pracovníků

z podobných realizací, časový harmonogram a další okolnosti související s realizací. Na základě těchto hodnocení lze učinit rozhodnutí zda, je společnost schopna a ochotna projekt realizovat.

V průběhu tohoto hodnocení je zahájen dialog se zákazníkem a jsou komunikovány základní požadavky na systém a právě benefity, které si realizací projektu jeho zadavatel slibuje. Na základě požadavků je realizátor schopen identifikovat nutné role a schopnosti členů týmu, který by projekt měl realizovat. Interně ve společnosti nastává prioritizace projektu mezi ostatními projekty, identifikací rolí a kontaktů na jednotlivé klíčové uživatele zákazníka. Očekávané přínosy pro zadavatele jsou pak transformovány do cílů projektu a jeho priority. Priorita je definována na základě financí, času a rozsahu implementace, kdy jeden rozměr je stanoven jako primární a ostatní jako sekundární. Například primárním rozměrem projektu bude čas (předání projektu je nutno učinit do určitého data bez možnosti odložení). Tímto se tedy sekundární rozměry projektu stávají obsah realizace (nemusíme realizovat veškeré funkcionality) a finance (projekt může překročit rozpočet, aby bylo dosaženo úplné implementace v daném čase).

Veškeré tyto aspekty nastávajícího projektu jsou shrnuty v dokumentu Direktiva projektu (viz. přílohy A .3), který je následně prezentován vedení společnosti a je učiněno rozhodnutí o započetí přípravných prací směřující k realizaci projektu.

Jako podpora těchto aktivit ve společnosti ELVAC SOLUTIONS jsou využity nástroje společnosti Atlassian JIRA [6] a Confluence [7]. V systému JIRA jsou evidovány hodiny provedené v této fázi projektu a práce, která byla vykonána v podobě tiketů. V systému Confluence jsou evidovány záznamy jednání se zákazníkem, nabídky, požadavky a specifikace zákazníka (viz. obrázek 9 a 10) a direktiva projektu. V systému Confluence jsou jako podpora této fáze připraveny šablony pro sběr požadavků, šablony pro záznamy a organizaci jednání se zákazníkem, direktivy projektu a jejich agregace přehledně na stránkách privátního prostoru projektu v Confluence. Veškerá funkcionalita těchto projektových prostorů využívá vlastní funkcionality Confluence a také různá rozšíření vlastní nebo třetích stran. Snadná integrace systémů JIRA a Confluence umožňuje vytvářet tikety do systému JIRA pouze označením textu na stránce v Confluence. Toho se hlavně využívá při definici úkolů pro členy týmu přímo z požadavků zákazníka (viditelné na obrázku 9) nebo záznamu z jednání. Takto je vytvořeno spojení každého požadavku nebo úkolu s konkrétním tiketem v systému JIRA, jeho řešitelem, časovým odhadem, způsobem řešení, verzí, odvedenou prací a dalšími informacemi (více o systému JIRA je uvedeno dále v textu). Tímto jsou také splněny procesní náležitosti normy ISO 12207, kterou je společnost ELVAC SOLUTIONS certifikována. Vlastní prostor projektu je generován automaticky pomocí interního nástroje společnosti a je takto zaručeno, že každý projekt bude splňovat veškeré definované náležitosti vycházející s normy ISO a interních směrnic.

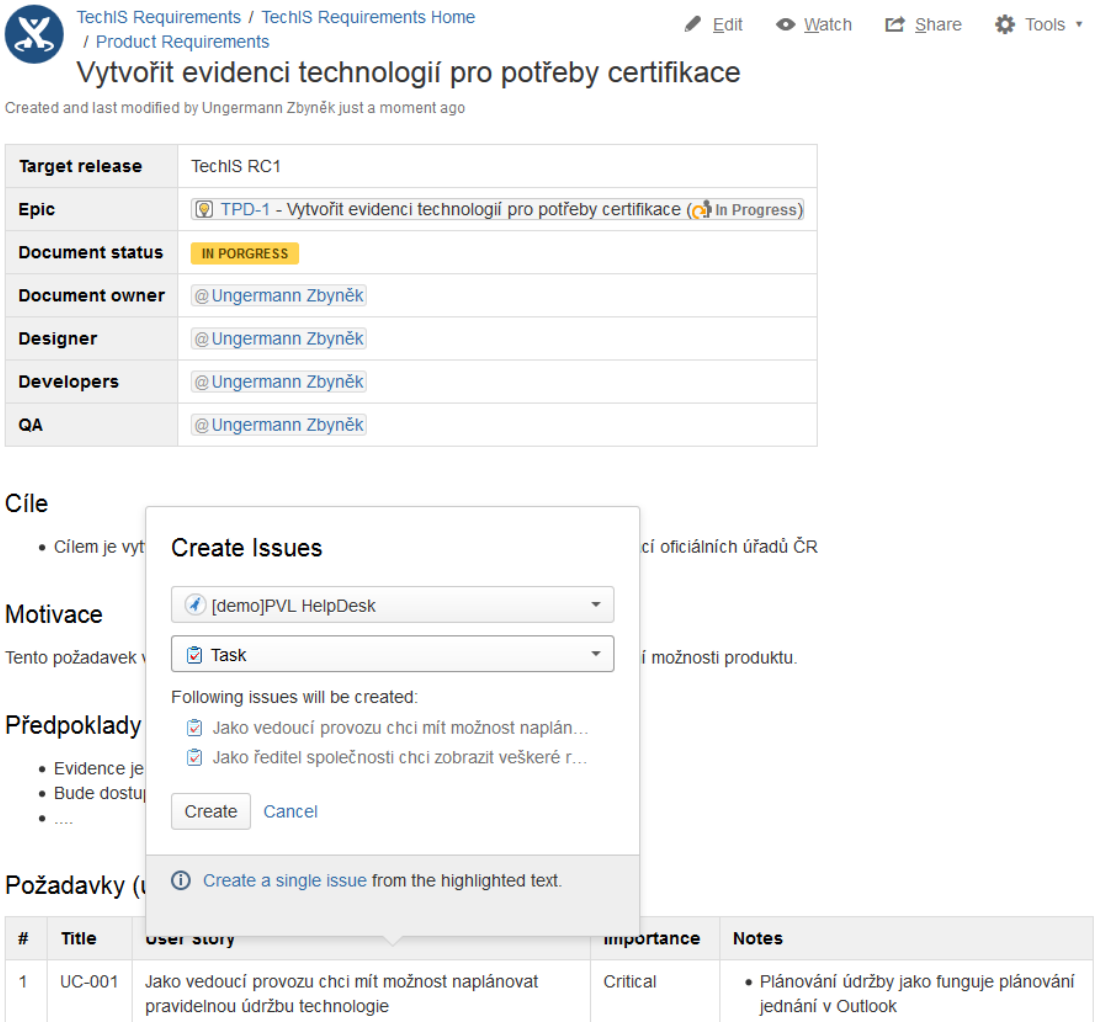


Figure 9: Stránka s analýzou požadavku zákazníka

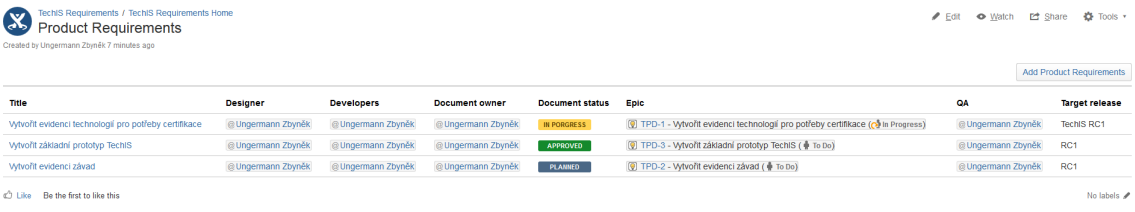


Figure 10: Přehledová stránka všech požadavků řešení

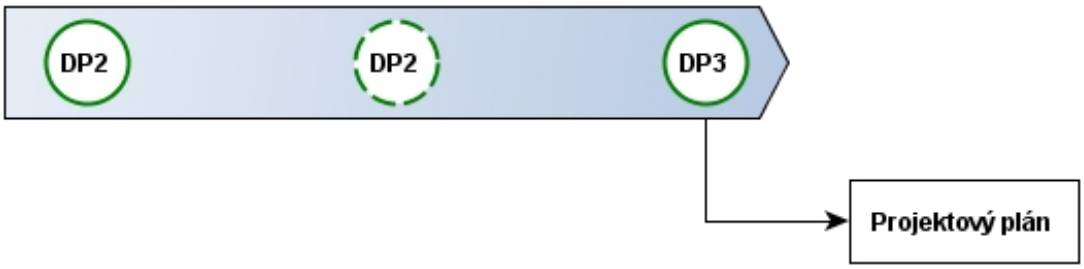


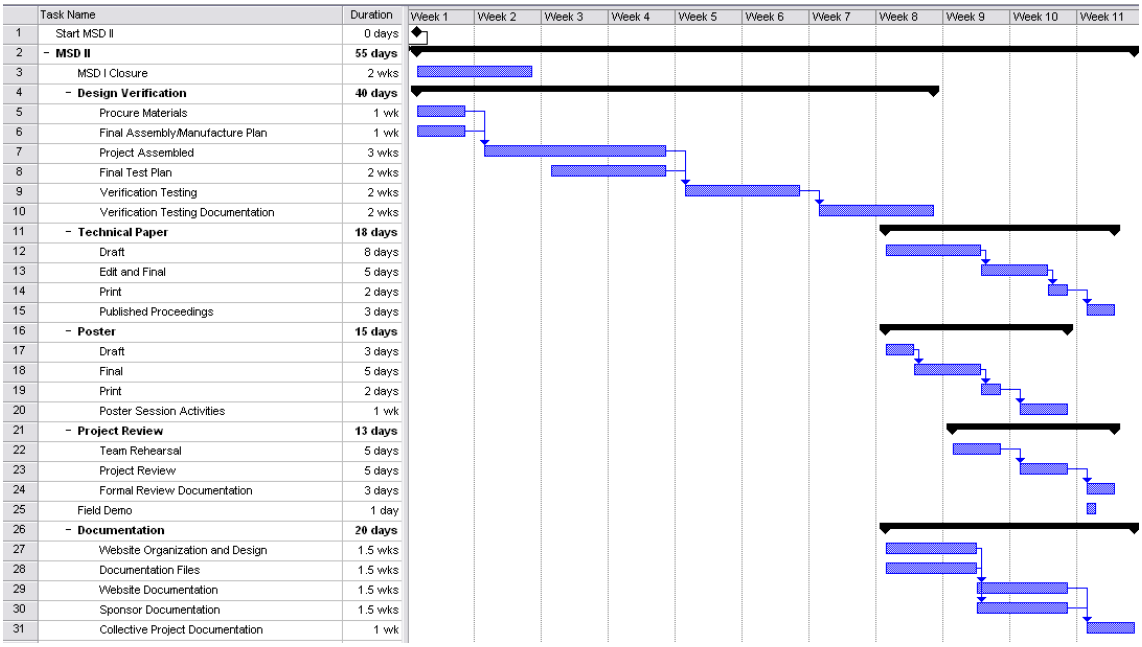
Figure 11: Projektový plán jako výstup fáze plánování projektu

7 .3 Plánování projektu

Po rozhodnutí o pokračování v projektu a schválení direktivy projetu, nastává detailnější plánování prací na projektu. Výsledkem této fáze je projektový plán. Tento dokument vychází z direktivy projektu a je rozšířen o definici rolí a ustavení řídící skupiny projektu (reprezentace zákazníka a realizátora projektu), analýzu rizik, jednotlivé milníky a akceptační kritéria, pracovní postupy a metody, způsob komunikace a sdílení informací a finanční zázemí projektů a odhadované náklady.

Milníky a obsah prací v projektu je definován pomocí WBS (Work Breakdown Structure) v nástroji Excel nebo Microsoft Project (obrázek 12). WBS slouží také k identifikaci kritické cesty v grafu a tedy identifikaci prioritních úloh. Ke každé části realizace zadání je přiřazen časový odhad náročnosti, odpovědný pracovník a další role nutné k úspěšnému splnění zadání, termín a akceptační kritéria. Veškerá tato dokumentace pokračuje v prostoru projektu v Confluence a pro zadávání úkolů je opět využit systém JIRA. V této fázi také dochází k prioritizaci požadavků zákazníka a detailnějšímu rozpracování požadavků s nejvyšší prioritou (tyto budou řešeny v rámci první iterace realizace). K definici a rozpracování požadavků je opět využito šablon v systému Confluence a ke specifikaci konkrétních úkolů je využita již zmíněná integrace se systémem JIRA.

Schválený rozvrh prací řídící skupinou projektu je transformován do systému JIRA pomocí verzí systému, které definují jednotlivé milníky projektu a pomocí komponent projektu, které specifikují oblasti řešení (například databáze, notifikace, GUI a podobné). JIRA následně umožňuje filtrovat pomocí takto zadaných verzí a komponent a usnadňuje orientaci v projektu pro vedení projektu a komunikaci s řídící skupinou projektu. Lze jednoduše sledovat odpracovaný čas, splněné úkoly a úkoly, které ještě chybí dokončit. V systému JIRA pomocí nástroje TEMPO jsou pak definovány alokace jednotlivých rolí a pracovníků společnosti v rámci verze, konkrétního úkolu nebo komponenty. Lze tak ihned identifikovat možné konflikty a rizika v alokaci pracovníků a například upravit plán projektu, tak aby bylo riziko odstraněno. Tento systém, také umožňuje pracovníkům reagovat na jim přidělenou práci a případně říci, že se na tento úkol necítí a tím opět umožnit včasnou reakci na možné problémy.



	Task Name	Duration
1	Start MSD II	0 days
2	**MSD II**	**55 days**
3	MSD I Closure	2 wks
4	**Design Verification**	**40 days**
5	Procure Materials	1 wk
6	Final Assembly/Manufacture Plan	1 wk
7	Project Assembled	3 wks
8	Final Test Plan	2 wks
9	Verification Testing	2 wks
10	Verification Testing Documentation	2 wks
11	**Technical Paper**	**18 days**
12	Draft	8 days
13	Edit and Final	5 days
14	Print	2 days
15	Published Proceedings	3 days
16	**Poster**	**15 days**
17	Draft	3 days
18	Final	5 days
19	Print	2 days
20	Poster Session Activities	1 wk
21	**Project Review**	**13 days**
22	Team Rehearsal	5 days
23	Project Review	5 days
24	Formal Review Documentation	3 days
25	Field Demo	1 day
26	**Documentation**	**20 days**
27	Website Organization and Design	1.5 wks
28	Documentation Files	1.5 wks
29	Website Documentation	1.5 wks
30	Sponsor Documentation	1.5 wks
31	Collective Project Documentation	1 wk

Figure 12: Work Breakdown Structure (WBS) v nástroji Microsoft Project

Projektový plán je, po této definici projektu na straně realizátora, prezentován řídící skupině projektu, která jej formálně schválí. Následně dojde k detailnímu plánu následující iterace vývoje a jeho schválení k implementaci.

Jako podpůrný nástroj je opět využito systému Confluence. V této fázi jde hlavně o schvalování formálních zápisů z jednání. Vydávaní oficiálních verzí dokumentů pomocí exportního nástroje ScrollOffice a předem definovaných šablon a metadat. Systém Confluence přehledně umožňuje verzování obsahu a tak je vždy dokladatelné kdo a jakou změnu provedl. Pro oficiální schválení dokumentu je využito rozšíření Confluence a nástroj pro schvalování obsahu nebo standardním podpisem exportovaného dokumentu. Exportovaný dokument je pak přiložen ke stránce, kde je opět verzován o případné změny. Verzování, prokazatelné seznámení a schválení dokumentů opět splňuje nároky normy ISO 12207 [10] a je plně automatizované včetně oznámení o změnách, o požadavku na schválení a vlastního schválení dokumentu.

Posledním krokem je vytvoření struktury projektu nástroji pro správu zdrojových kódů - VisualSVN. Tato struktura je pevně definována směrnicemi ELVAC SOLUTIONS a o její strukturu se opírají nástroje Continuous Integration.

7 .4 Realizace projektu

Vlastní realizace je nejdelší a nejdůležitější fází projektu. Zde dochází k vývoji software a naplnění očekávání zákazníka. Realizace je rozdělena do dílčích procesů v rámci jedné iterace (vývojového cyklu) - plánování, implementace, představení a zhodnocení (retrospektiva). Jako nástroje podporující tyto procesy je využito opět placených nástrojů JIRA a Confluence a dále pak VisualSVN pro SCM, CruiseControl.NET [3] pro CI, powershell [3], NAnt [3], MSBuild [3] a další nástroje. Jejich konfiguraci, nastavení se zde pokusím nastínit a veškeré zmíněné skripty a konfigurace jsou součástí této práce jako přílohaa A .3.

7 .4.1 Příprava na realizaci a prototypování

V rámci procesního řízení lze započít implementaci ještě před vlastním schválením projektového plánu. Tohoto se využívá hlavně pro přípravu a představení prototypu řešení pro zákazníka, které implementuje podporu vybraného procesu nebo požadavku zákazníka. Tvorba prototypu (nebo úvodní struktury řešení) je v ELVAC SOLUTIONS opět plně automatizována a jeho vytvoření již definuje veškeré náležitosti, které jsou potřeba pro Continuous Integration a které budou popsány dále. Základem pro prototyp (a budoucí řešení) je interní projekt, který obsahuje veškeré sdílené knihovny, moduly a obecnou funkcionalitu. Pomocí powershell skriptu (viz příloha A .3) je pak tato šablona projektu převedena na řešení zákazníka. Tento skript přejmenuje veškeré jmenné prostory řešení, reference, databázové nastavení, konfiguraci nasazení a sestavení a další jména vytvoří tak nové řešení, které je vloženo do SVN. Spolu s tímto projektem jsou v SVN uloženy konfigurace pro podporu Continuous Integration na integračním serveru, nastavení sestavení, konfigurace pro sběr a kontrolu KPI a metrik zdrojových kódů, podpůrné skripty a další, které jsou transformovány spolu s projektem a je tak vytvořena úvodní struktura projektu s podporou Continuous Integration. Posledním rokem při tvorbě nových řešení je jejich převod na kódování UTF-8, které zajišťuje správné formátování znaků v řešení.

7 .4.2 Plánování iterace

Plánování obsahu probíhá vždy na začátku iterace společně se zákazníkem. Dojde k prioritizaci úkolů v projektovém zásobníku a dle priorit jsou zvoleny funkcionality k implementaci. Dojde k odhadu náročnosti, detailnímu popisu úkolu, definici tiketů v systému JIRA, odpovědných řešitelů a další nutné náležitosti. Tento způsob práce je založen na metodologii SCRUM popsané v úvodní části dokumentu.

Při plánování iterace jsou v systému JIRA označeny nebo vytvořeny veškeré úkoly, které jsou určeny k implementaci v dané iteraci. Pro označení těchto úkolů se používá verze v systému JIRA. Veškeré plánované úkoly tak sdílí verzi, která byla definována v projektovém plánu. Verze má nastaven svůj počátek a konec a pro vedení projektu je otázkou jednoho kliknutí v systému JIRA, aby v každém okamžiku zjistil, jak vypadá postup prací a mohl podniknout případné nápravné opatření. V rámci plánování úkolů dochází si-

multánně k plánovaní alokací jednotlivých pracovníků v dané iteraci a získaní obecného povědomí o obsahu a cílech iterace. Při plánování se určují také priority a závislosti jednotlivých úkolů navzájem. K tomuto je opět využito nativní funkcionality a podpory v systému JIRA. Tímto jsou v podstatě i postupy prací v rámci iterace a body zájmu a milníky v rámci iterace. Toto pomáhá minimalizovat rizika i na úrovni jednotlivých iterací.

7 .4.3 Implementace iterace

Po plánu dochází k vlastní implementaci. V rámci procesů se zde využívá kombinace nástrojů, tak aby bylo dosaženo co nejvyšší úrovně automatizace a podpory pro vlastní vývoj [1] [2]. Nástroje a jejich roli v procesu implementace se zde pokusím postupně popsat.

7 .4.3.1 JIRA a Confluence

Jak již bylo několikrát zmíněno v systému JIRA jsou definovány a uloženy veškeré úkoly dané iterace. Integrace systémů JIRA a Confluence umožňuje automatické a jednoduché mapování jednotlivých úkolů na požadavky zákazníka a pro kontrolu. Obsah iterace (respektive verze v systému JIRA) je přehledně prezentován na stránkách projektu v Confluence.

Další rolí JIRA je v tomto procesu také evidence odvedené práce. Toto je nejdůležitější pro sledování projektových KPI v průběhu projektu. JIRA dále definuje postupy prací a návaznosti (pomocí priorit a propojení úkolů ve fázi plánování) a slouží ke komunikaci mezi členy projektového týmu a zákazníkem.

Pro podporu vývoje byl v rámci mého působení v ELVAC SOLUTIONS nastaven automatický pracovní postup (viz obrázek 13), který automatizuje většinu prací na straně členů vývojového týmu a zároveň plně podporuje procesy a doporučení normy ISO. Toto jednoduché workflow (pracovní postup) obsahuje pouze 4 základní stavy - Otevřeno, V řešení, Revize, Uzavřeno. V rámci tohoto postupu je řešiteli přiřazen úkol ve stavu **Otevřeno**. V tomto stavu není povoleno zadávání práce k úkolu, aby nedocházelo k rozdílům v metrikách pro KPI. Řešitel započne práci na řešení tím, že tiket přesune do stavu **V řešení**. Po vyřešení je tiket přesunut do stavu **Revize** a je přiřazen odpovědné osobě ke kontrole odvedené práce nebo k otestování uvedené funkcionality. Po ověření dojde k uzavření úkolu. V rámci tohoto pracovního postupu jsou automaticky kontrolovány různé náležitosti tiketu (verze, záznam práce, komentáře, způsob vyřešení a podobně) a kontrola oprávnění prováděných akcí nad tiketem. Kopie exportovaného workflow je přiložena jako příloha A .3 této práce.

7 .4.3.2 CruiseControl.NET

Jako integrační server při implementaci Continuous Integration v ELVAC SOLUTIONS

Figure 13: JIRA pracovní postup

byl zvolen nástroj CruiseControl.NET [3] [8]. Pro každý projekt vyvíjený v ELVAC SOLUTIONS jsou definovány základní integrační úlohy na tomto serveru:

- Continuous Integration úloha
- Úloha pro generování dokumentace
- Úloha pro nasazení řešení
- Úloha pro vydání instalátoru pro řešení

Server v pravidelných intervalech (konfigurovaných na úrovni každé úlohy) kontroluje úložiště zdrojových kódů a v případě změny vykoná akce nastavené v konfiguraci každé úlohy. Integrační úlohy jsou konfigurovány v souborech XML a takto jsou načteny integračním serverem. XML konfigurace využívá sdílených nastavení, která jsou obecná pro všechny projekty v ELVAC SOLUTIONS (součástí přílohy A .3). Tyto nastavení slouží k definici:

- Emailové notifikace při selhání sestavení pro uživatele, kteří jsou za ně odpovědní
- Oprávnění na integrační projekty - například definice pozice release manager, který může spustit vydání aplikace z integračního serveru
- Filtry modifikací, které spustí automatické sestavení
- Definice, konfiguraci a operace SVN severu v případě nového sestavení nebo neúspěšného sestavení

- Způsob publikování výsledků a publikované informace na stránkách integračního serveru
- Dynamické parametry pro sestavení aplikace a dokumentace

Zmíněné dynamické parametry minimalizují počet úloh na build serveru a tak zjednodušují údržbu úloh a konfigurací. Pomocí těchto parametrů je úlohu možno konfigurovat až před jejím spuštěním. Například v případě generování dokumentace, lze zvolit jaký typ dokumentace má být vydán - programátorská, uživatelská v pdf nebo webová v pdf (obrázek 14). V případě release řešení je možné specifikovat o jaký typ release se

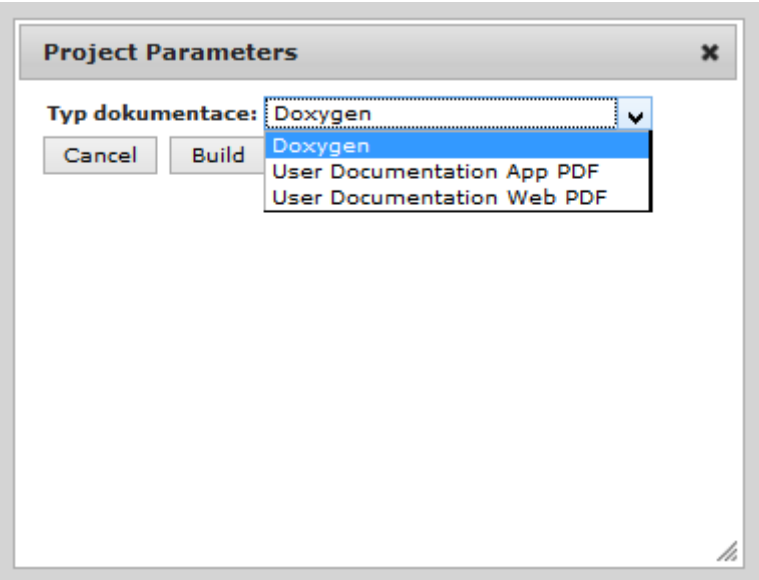


Figure 14: CruiseControl.NET dynamické nastavení dokumentace

jedná (lokální nebo zákaznický), zda se v průběhu release má vytvořit nové schéma v databázi, zda-li se mají vytvořit i soubory pro debug aplikace, zda-li se má vytvořit štítek v SCM pro daný release (tento štítek pak slouží k rychlému návratu k produkčním verzím řešení u zákazníka v případě problémů) a různé kombinace těchto nastavení (obrázek 15).

Emailové notifikace byly nastaveny, tak aby členové týmu dostávali pouze důležité informace a aby nedocházelo k přehlcení vývojářů emaily a tím pádem k degradaci rychlého a responsivního způsobu práce propagovaného právě pomocí metod Continuous Integration [1] a SCRUM. Vývojáři jsou tak upozorněni pouze v případě, že sestavení aplikace neproběhne úspěšně. Ostatní emaily jsou doručovány pouze správci integračního serveru. Dodatečné informace jsou přístupné pro všechny členy týmu pomocí webového rozhraní integračního serveru.

Úlohy pro Continuous Integration každého řešení provádí stejné, pevně definované aktivity, které slouží k určení kvality kódu a slouží jako hlavní identifikátor pro vývojáře,

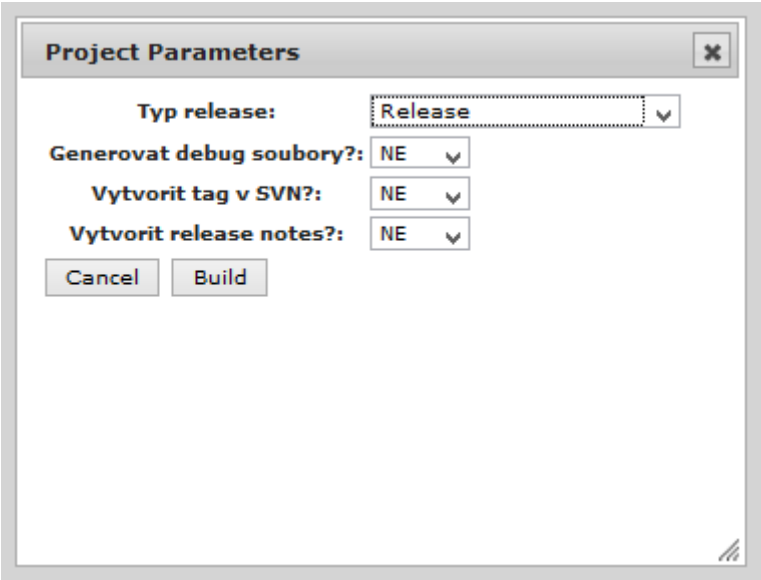


Figure 15: CruiseControl.NET dynamické nastavení release aplikace

zda jimi odvedená práce je akceptována či nikoliv. V rámci těchto úloh jsou prováděny testy řešení a výpočet pokrytí kódu testy, kontrola kvalitativních metrik, kontrola změn v databázi a databázovém schématu, kontrola referencí a další operace nutné k identifikaci správnosti odvedené práce z pohledu integrace řešení s dalšími celky.

Konfigurace těchto úloh na integračním serveru v podstatě pouze definuje jméno a identifikaci projektu, jeho integrační frontu, pracovní adresář a adresář pro uložení výsledku integrační úlohy, úložiště projektu v SVN a typ úlohy (viz A .1). Veškeré ostatní parametry jsou převzaty z obecné konfigurace integračního serveru. Naopak vlastní úkony jako sestavení, výpočet metrik, provádění unit testů, nasazení a další, jsou definovány na úrovni jednotlivých projektů pomocí nástroje NAnt. V NAnt skriptech je také provedena konfigurace umístění balíčků aplikace, umístění webového rozhraní a další konfigurace sestavení, tak aby nemusela být součástí konfigurace úloh na integračním serveru. Takto je dosaženo nezávislosti úloh na serveru na jejich vlastním obsahu a usnadňuje se tak jejich automatická konfigurace a jednoduchá opakovatelnost a integrace s ostatními projekty.

Během sestavení řešení jsou vykonávány další aktivity, které slouží pro sběr metrik a analýzu KPI. Veškeré tyto aktivity ukládají svoje výstupy do XML souborů, které jsou na konci integrační úlohy spojeny do jednoho souboru. Pomocí XSL transformací tohoto souboru je pak možné přehledně prezentovat výsledky sestavení, výsledky testů (Obrázek 17), metriky a KPI přehledně v rámci webového rozhraní CruiseControl.NET. Transformace tohoto souboru se využívá i při generování notifikací pro členy týmu. Instalace CruiseControl.NET obsahuje předdefinované soubory transformací, ale pro potřeby prezentace, vzhledem ke specifickým úkolům při vývoji v DevExpress bylo nutné v rámci

Test Run: ccnet_test

Summary

Designated outcome : **Completed**

Number of Tests	Passed	Failed	Inconclusive	Aborted	Timeout	Started at	Stopped at
56	56	0	0	0	0	25-03-2014 10:56:09	25-03-2014 10:56:37

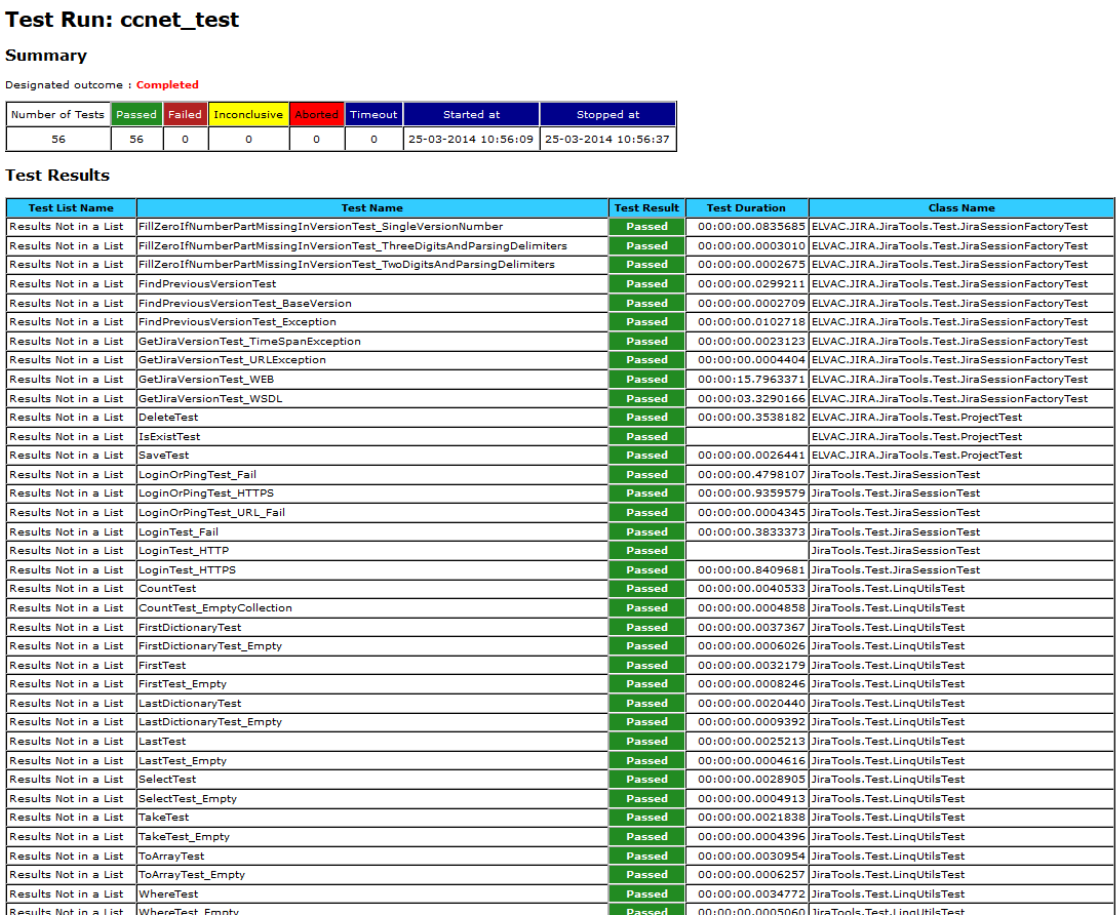
Test Results

Test List Name	Test Name	Test Result	Test Duration	Class Name
Results Not in a List	FillZeroIfNumberPartMissingInVersionTest_SingleVersionNumber	Passed	00:00:00.0835685	ELVAC.JIRA.JiraTools.Test.JiraSessionFactoryTest
Results Not in a List	FillZeroIfNumberPartMissingInVersionTest_ThreeDigitsAndParsingDelimiters	Passed	00:00:00.0003010	ELVAC.JIRA.JiraTools.Test.JiraSessionFactoryTest
Results Not in a List	FillZeroIfNumberPartMissingInVersionTest_TwoDigitsAndParsingDelimiters	Passed	00:00:00.0002675	ELVAC.JIRA.JiraTools.Test.JiraSessionFactoryTest
Results Not in a List	FindPreviousVersionTest	Passed	00:00:00.0299211	ELVAC.JIRA.JiraTools.Test.JiraSessionFactoryTest
Results Not in a List	FindPreviousVersionTest_BaseVersion	Passed	00:00:00.0002709	ELVAC.JIRA.JiraTools.Test.JiraSessionFactoryTest
Results Not in a List	FindPreviousVersionTest_Exception	Passed	00:00:00.0102718	ELVAC.JIRA.JiraTools.Test.JiraSessionFactoryTest
Results Not in a List	GetJiraVersionTest_TimeSpanException	Passed	00:00:00.0023123	ELVAC.JIRA.JiraTools.Test.JiraSessionFactoryTest
Results Not in a List	GetJiraVersionTest_URLException	Passed	00:00:00.0004404	ELVAC.JIRA.JiraTools.Test.JiraSessionFactoryTest
Results Not in a List	GetJiraVersionTest_WEB	Passed	00:00:15.7963371	ELVAC.JIRA.JiraTools.Test.JiraSessionFactoryTest
Results Not in a List	GetJiraVersionTest_WSDL	Passed	00:00:03.3290166	ELVAC.JIRA.JiraTools.Test.JiraSessionFactoryTest
Results Not in a List	DeleteTest	Passed	00:00:00.3538182	ELVAC.JIRA.JiraTools.Test.ProjectTest
Results Not in a List	IsExistTest	Passed		ELVAC.JIRA.JiraTools.Test.ProjectTest
Results Not in a List	SaveTest	Passed	00:00:00.0026441	ELVAC.JIRA.JiraTools.Test.ProjectTest
Results Not in a List	LoginOrPingTest_Fail	Passed	00:00:00.4798107	JiraTools.Test.JiraSessionTest
Results Not in a List	LoginOrPingTest_HTTPS	Passed	00:00:00.9359579	JiraTools.Test.JiraSessionTest
Results Not in a List	LoginOrPingTest_URL_Fail	Passed	00:00:00.0004345	JiraTools.Test.JiraSessionTest
Results Not in a List	LoginTest_Fail	Passed	00:00:00.3833373	JiraTools.Test.JiraSessionTest
Results Not in a List	LoginTest_HTTP	Passed		JiraTools.Test.JiraSessionTest
Results Not in a List	LoginTest_HTTPS	Passed	00:00:00.8409681	JiraTools.Test.JiraSessionTest
Results Not in a List	CountTest	Passed	00:00:00.0040533	JiraTools.Test.LinqUtilsTest
Results Not in a List	CountTest_EmptyCollection	Passed	00:00:00.0004858	JiraTools.Test.LinqUtilsTest
Results Not in a List	FirstDictionaryTest	Passed	00:00:00.0037367	JiraTools.Test.LinqUtilsTest
Results Not in a List	FirstDictionaryTest_Empty	Passed	00:00:00.0006026	JiraTools.Test.LinqUtilsTest
Results Not in a List	FirstTest	Passed	00:00:00.0032179	JiraTools.Test.LinqUtilsTest
Results Not in a List	FirstTest_Empty	Passed	00:00:00.0008246	JiraTools.Test.LinqUtilsTest
Results Not in a List	LastDictionaryTest	Passed	00:00:00.0020440	JiraTools.Test.LinqUtilsTest
Results Not in a List	LastDictionaryTest_Empty	Passed	00:00:00.0009392	JiraTools.Test.LinqUtilsTest
Results Not in a List	LastTest	Passed	00:00:00.0025213	JiraTools.Test.LinqUtilsTest
Results Not in a List	LastTest_Empty	Passed	00:00:00.0004616	JiraTools.Test.LinqUtilsTest
Results Not in a List	SelectTest	Passed	00:00:00.0028905	JiraTools.Test.LinqUtilsTest
Results Not in a List	SelectTest_Empty	Passed	00:00:00.0004913	JiraTools.Test.LinqUtilsTest
Results Not in a List	TakeTest	Passed	00:00:00.0021838	JiraTools.Test.LinqUtilsTest
Results Not in a List	TakeTest_Empty	Passed	00:00:00.0004396	JiraTools.Test.LinqUtilsTest
Results Not in a List	ToArrayTest	Passed	00:00:00.0030954	JiraTools.Test.LinqUtilsTest
Results Not in a List	ToArrayTest_Empty	Passed	00:00:00.0006257	JiraTools.Test.LinqUtilsTest
Results Not in a List	WhereTest	Passed	00:00:00.0034772	JiraTools.Test.LinqUtilsTest
Results Not in a List	WhereTest_Empty	Passed	00:00:00.0005060	JiraTools.Test.LinqUtilsTest

Figure 16: Prezentace výsledků testů na integračním serveru

ELVAC SOLUTIONS definovat vlastní transformace tak, aby vývojový tým v každém oznámení o selhání sestavení dostal informace, které potřebuje. Takto je opět dosaženo co nejrychlejší reakce na vzniklé problémy a umožňuje rychlou identifikaci a opravu problému.

CruiseControl.NET také umožňuje přehledně zobrazit statistiky jako je počet ses-



Test Run: ccnet_test

Summary

Designated outcome : **Completed**

Number of Tests	Passed	Failed	Inconclusive	Aborted	Timeout	Started at	Stopped at
56	56	0	0	0	0	25-03-2014 10:56:09	25-03-2014 10:56:37

Test Results

Test List Name	Test Name	Test Result	Test Duration	Class Name
Results Not in a List	FillZeroIfNumberPartMissingInVersionTest_SingleVersionNumber	Passed	00:00:00.0835685	ELVAC.JIRA.JiraTools.Test.JiraSessionFactoryTest
Results Not in a List	FillZeroIfNumberPartMissingInVersionTest_ThreeDigitsAndParsingDelimiters	Passed	00:00:00.0003010	ELVAC.JIRA.JiraTools.Test.JiraSessionFactoryTest
Results Not in a List	FillZeroIfNumberPartMissingInVersionTest_TwoDigitsAndParsingDelimiters	Passed	00:00:00.0002675	ELVAC.JIRA.JiraTools.Test.JiraSessionFactoryTest
Results Not in a List	FindPreviousVersionTest	Passed	00:00:00.0299211	ELVAC.JIRA.JiraTools.Test.JiraSessionFactoryTest
Results Not in a List	FindPreviousVersionTest_BaseVersion	Passed	00:00:00.0002709	ELVAC.JIRA.JiraTools.Test.JiraSessionFactoryTest
Results Not in a List	FindPreviousVersionTest_Exception	Passed	00:00:00.0102718	ELVAC.JIRA.JiraTools.Test.JiraSessionFactoryTest
Results Not in a List	GetJiraVersionTest_TimeSpanException	Passed	00:00:00.0023123	ELVAC.JIRA.JiraTools.Test.JiraSessionFactoryTest
Results Not in a List	GetJiraVersionTest_URLException	Passed	00:00:00.0004404	ELVAC.JIRA.JiraTools.Test.JiraSessionFactoryTest
Results Not in a List	GetJiraVersionTest_WEB	Passed	00:00:15.7963371	ELVAC.JIRA.JiraTools.Test.JiraSessionFactoryTest
Results Not in a List	GetJiraVersionTest_WSDL	Passed	00:00:03.3290166	ELVAC.JIRA.JiraTools.Test.JiraSessionFactoryTest
Results Not in a List	DeleteTest	Passed	00:00:00.3538182	ELVAC.JIRA.JiraTools.Test.ProjectTest
Results Not in a List	IsExistTest	Passed		ELVAC.JIRA.JiraTools.Test.ProjectTest
Results Not in a List	SaveTest	Passed	00:00:00.0026441	ELVAC.JIRA.JiraTools.Test.ProjectTest
Results Not in a List	LoginOrPingTest_Fail	Passed	00:00:00.4798107	JiraTools.Test.JiraSessionTest
Results Not in a List	LoginOrPingTest_HTTPS	Passed	00:00:00.9359579	JiraTools.Test.JiraSessionTest
Results Not in a List	LoginOrPingTest_URL_Fail	Passed	00:00:00.0004345	JiraTools.Test.JiraSessionTest
Results Not in a List	LoginTest_Fail	Passed	00:00:00.3833373	JiraTools.Test.JiraSessionTest
Results Not in a List	LoginTest_HTTP	Passed		JiraTools.Test.JiraSessionTest
Results Not in a List	LoginTest_HTTPS	Passed	00:00:00.8409681	JiraTools.Test.JiraSessionTest
Results Not in a List	CountTest	Passed	00:00:00.0040533	JiraTools.Test.LinqUtilsTest
Results Not in a List	CountTest_EmptyCollection	Passed	00:00:00.0004858	JiraTools.Test.LinqUtilsTest
Results Not in a List	FirstDictionaryTest	Passed	00:00:00.0037367	JiraTools.Test.LinqUtilsTest
Results Not in a List	FirstDictionaryTest_Empty	Passed	00:00:00.0006026	JiraTools.Test.LinqUtilsTest
Results Not in a List	FirstTest	Passed	00:00:00.0032179	JiraTools.Test.LinqUtilsTest
Results Not in a List	FirstTest_Empty	Passed	00:00:00.0008246	JiraTools.Test.LinqUtilsTest
Results Not in a List	LastDictionaryTest	Passed	00:00:00.0020440	JiraTools.Test.LinqUtilsTest
Results Not in a List	LastDictionaryTest_Empty	Passed	00:00:00.0009392	JiraTools.Test.LinqUtilsTest
Results Not in a List	LastTest	Passed	00:00:00.0025213	JiraTools.Test.LinqUtilsTest
Results Not in a List	LastTest_Empty	Passed	00:00:00.0004616	JiraTools.Test.LinqUtilsTest
Results Not in a List	SelectTest	Passed	00:00:00.0028905	JiraTools.Test.LinqUtilsTest
Results Not in a List	SelectTest_Empty	Passed	00:00:00.0004913	JiraTools.Test.LinqUtilsTest
Results Not in a List	TakeTest	Passed	00:00:00.0021838	JiraTools.Test.LinqUtilsTest
Results Not in a List	TakeTest_Empty	Passed	00:00:00.0004396	JiraTools.Test.LinqUtilsTest
Results Not in a List	ToArrayTest	Passed	00:00:00.0030954	JiraTools.Test.LinqUtilsTest
Results Not in a List	ToArrayTest_Empty	Passed	00:00:00.0006257	JiraTools.Test.LinqUtilsTest
Results Not in a List	WhereTest	Passed	00:00:00.0034772	JiraTools.Test.LinqUtilsTest
Results Not in a List	WhereTest_Empty	Passed	00:00:00.0005060	JiraTools.Test.LinqUtilsTest

Figure 17: Prezentace výsledků testů na integračním serveru

tavení za den (viz. Obrázek 18), rychlost každého sestavení (viz. Obrázek 19), statistiku porušení pravidel FxCop (viz. Obrázek 20) a například souhrnné statistiky úspěšnosti sestavení (viz. Obrázek 21) daného projektu v rámci svého webového rozhraní. V rámci ELVAC SOLUTIONS integrační server spravuje přibližně 50 pravidelných úloh pro circa 15 projektů.

7 .4.3.3 NAnt

NAnt je nástroj pro definici sestavení .NET aplikací [3] [9]. NAnt dále obsahuje možnost

Build Summary Statistics

Day	Index	Last Build Label	Successful Build Count	Failed Build Count	Average Build Duration	Min Build Duration	Max Build Duration	Average Coverage	Test Count	Tests Passed	Test Failures	Tes
+ Fri Apr 18 2014	103	526	9	1	497.89	350	858	0	0	0	0	
+ Thu Apr 17 2014	102	517	16	2	438.81	336	624	0	0	0	0	
+ Wed Apr 16 2014	101	501	17	1	511.24	322	1196	0	0	0	0	
- Tue Apr 15 2014	100	484	10	1	404.4	334	562	0	0	0	0	

Build Label	Status	Start Time	Duration	Test Count	Test Failures	Test Ignored	Gendarme Defects	Fx Cop Warnings	Fx Cop Errors	Build Error Type	Build Error M
484	Success	2014-04-15 23:20:18	00:05:42	0	0	0	0	1900	863		
483	Success	2014-04-15 18:05:53	00:09:22	0	0	0	0	1900	863		
482	Success	2014-04-15 15:34:17	00:07:56	0	0	0	0	1847	859		
481	Success	2014-04-15 14:18:41	00:07:33	0	0	0	0	1847	859		
481	Failure	2014-04-15 14:04:20	00:03:01	0	0	0	0	0	0	NAnt.Core.BuildException	External Program Failed: c:\windc \v4.0.30319\msbuild.exe
480	Success	2014-04-15 10:34:02	00:06:46	0	0	0	0	1847	859		
479	Success	2014-04-15 09:50:04	00:05:54	0	0	0	0	1847	859		
478	Success	2014-04-15 09:00:53	00:06:11	0	0	0	0	1847	859		
477	Success	2014-04-15 08:54:03	00:05:43	0	0	0	0	1847	859		
476	Success	2014-04-15 07:46:26	00:05:34	0	0	0	0	1847	859		
475	Success	2014-04-15 07:32:13	00:06:43	0	0	0	0	1847	859		

Mon Apr 14 2014	99	474	6	1	391.83	330	484	0	0	0	0	

Build Label	Status	Start Time	Duration	Test Count	Test Failures	Test Ignored	Gendarme Defects	Fx Cop Warnings	Fx Cop Errors	Build Error Type	Build Error M
474	Success	2014-04-14 15:13:07	00:05:44	0	0	0	0	1847	859		
474	Failure	2014-04-14 14:55:53	00:06:38	0	0	0	0	0	0	NAnt.Core.BuildException	External Program Failed: D:\CCN \Artifac \out\ELVAC.TechIS.Tools.Databas code wa
473	Success	2014-04-14 13:17:14	00:08:04	0	0	0	0	1847	859		
472	Success	2014-04-14 12:41:24	00:06:42	0	0	0	0	1847	859		
471	Success	2014-04-14 10:40:39	00:05:32	0	0	0	0	1847	859		
470	Success	2014-04-14 10:34:02	00:05:30	0	0	0	0	1847	859		
469	Success	2014-04-14 10:17:29	00:07:39	0	0	0	0	1847	859		

+ Fri Apr 11 2014	98	468	13	5	391.08	319	549	0	0	0	0	
+ Thu Apr 10 2014	97	455	10	0	395.7	325	530	0	0	0	0	
+ Wed Apr 09 2014	96	445	18	2	465.5	293	811	0	0	0	0	
+ Tue Apr 08 2014	95	427	6	3	444.17	326	564	0	0	0	0	
+ Mon Apr 07 2014	94	421	8	0	368.75	306	446	0	0	0	0	

Figure 18: Prezentace statistik úlohy na integračním serveru

Build Duration

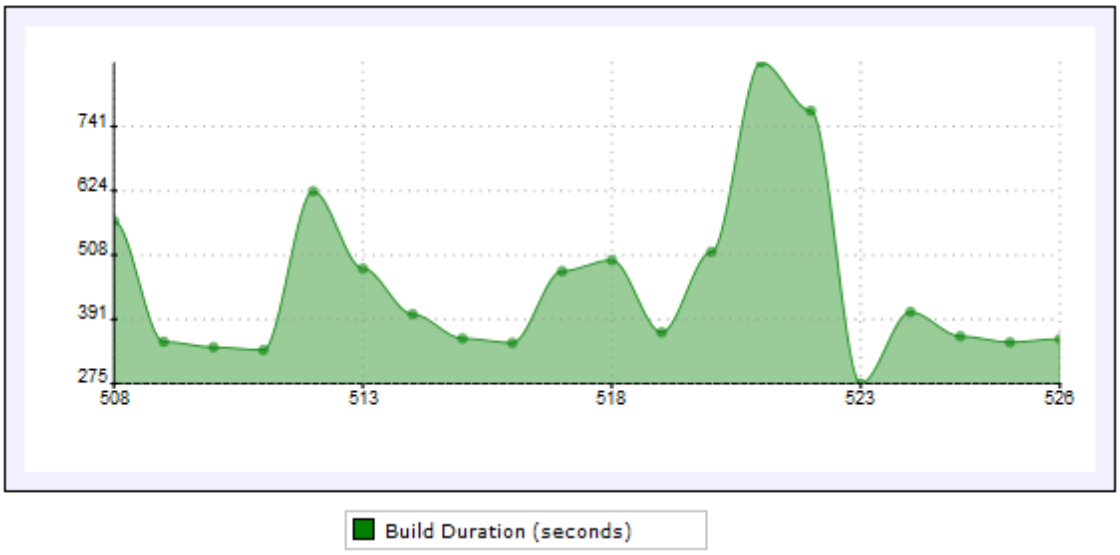


Figure 19: Prezentace rychlosti sestavení úlohy na integračním serveru

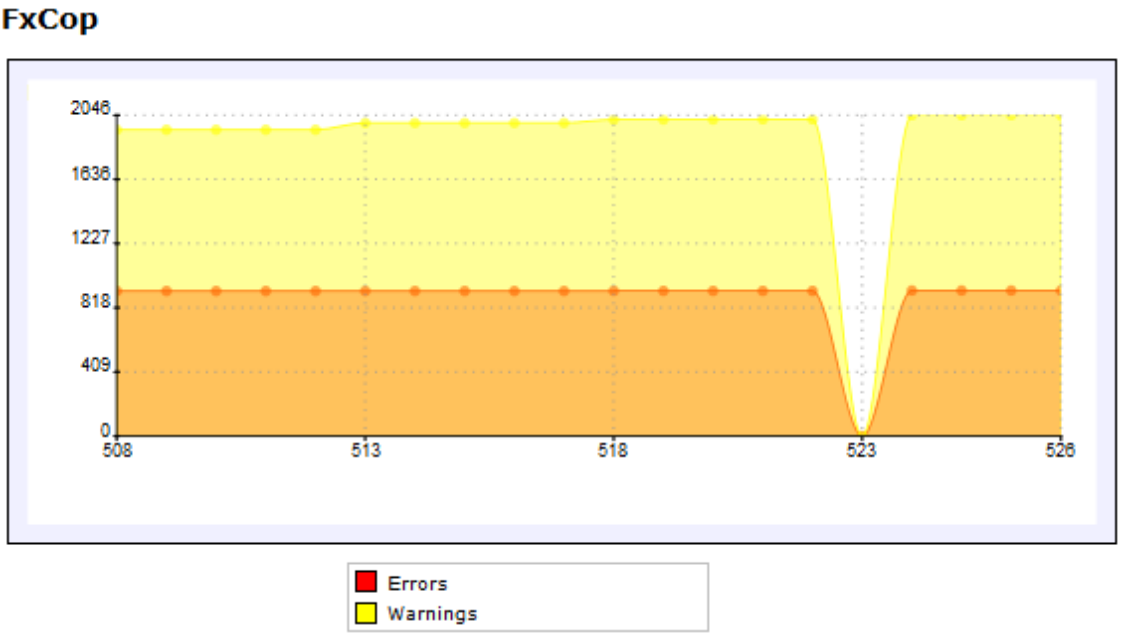


Figure 20: Výsledky analýzy FxCop na integračním serveru

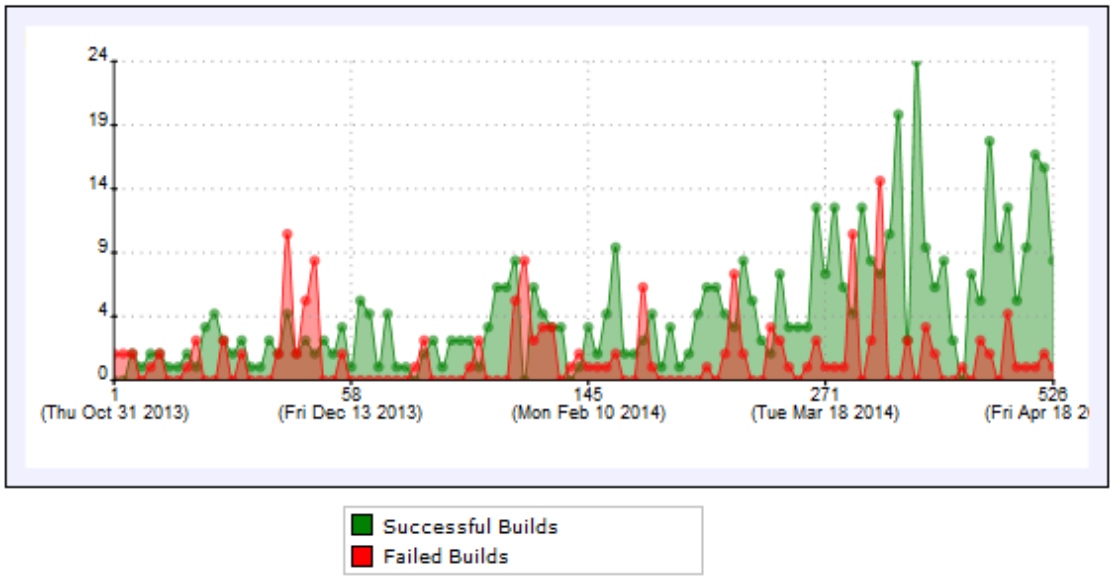


Figure 21: Úspěšnost sestavení úlohy na integračním serveru

rozšířit definici sestavení a další operace jako je kopírování souborů, vytváření adresářů, hromadné úpravy, volání externích programů a spoustu dalších funkcí a možností.

Pro správné funkce NAnt v prostředí Continuous Integration ve společnosti ELVAC SOLUTIONS byl definován standardní soubor nástrojů, které podporují jednotlivé aktivity skriptů NAnt. Tato sada nástrojů obsahuje powershell skripty, aplikace pro generování dokumentace DoxyGen a další nutné nástroje pro správnou funkcionalitu a podporu všech Continuous Integration aktivit na straně integračního serveru. Veškeré nástroje a skripty jsou součástí přílohy A .3 této práce.

Vlastní konfigurace sestavení a dalších úloh probíhá podle úloh NAnt (NAnt target), jejichž voláním jsou prováděny jednotlivé úkony nebo sestavy úkonů Continuous Integration. Úkolem této práce není popis definice těchto rozšíření a proto jsou jednotlivé soubory definující tyto aktivity součástí přílohy A .3 této práce . Oddělením nástrojů od vlastní implementace skriptů bylo dosaženo i důležitého faktoru nezávislosti sestavení na integračním serveru a umožňuje tak sestavení aplikace na počítačích členů vývojového týmu stejným způsobem jako k tomu dojde na integračním serveru. Toto podstatně zrychluje proces zpětné vazby a umožňuje vývojáři se ujistit o správnosti vyvíjeného artefaktu dříve než integrací na serveru.

Skripty jsou rozděleny, podle typů úloh, které vykonávají, do několika celků:

- Vydání aplikace
- Sestavení aplikace
- Kontrola kvality řešení
- Nasazení
- Vytvoření dokumentace
- a další

Jednotlivé vlastnosti integrace pro různé artefakty řešení (vlastní aplikace a instalátor) jsou pak konfigurovány v hlavních souborech, které obsahují pouze seznam základních proměnných a nastavení (viz. příloha A .2). Takto vznikla struktura souborů definující veškeré možné akce k sestavení a vydání řešení, které jsou vzájemně propojeny.

NAnt skripty v sobě integrují nástroje jako jsou powershell, MSTest, FxCop, StyleCop, operace se serverem SVN, komprimování souborů a další podpůrné nástroje [3].

7 .4.3.4 VisualSVN

Jako nástroj pro správu zdrojových kódů byl zvolen VisualSVN server. Nástroj obsahuje veškeré funkcionality pro řízení zdrojových kódů jako je vytváření a spojování vývojových větví, automatické akce před a po vložení nové verze zdrojových kódů a definici oprávnění

pro uživatele na úrovni jednotlivých projektů. Mimo podpory vlastního vývoje je role SVN serveru důležitá právě pro integrační server, kde slouží jako zdroj dat. Integrační server v pravidelných intervalech kontroluje definované projekty a v případě změny si stáhne ze serveru SVN poslední verzi a nad touto provede sestavení a veškeré další úkony Continuous Integration.

Na SVN serveru je nastavena akce, která před vložením kódu zkontroluje kódování zdrojových kódů a v případě, že jsou v jiném kódování než UTF-8 tento commit odmítne. Skript zajišťující tuto kontrolu je opět součástí přílohy A .3 této práce.

Struktura SVN byla pevně definována dle doporučení komunity SVN na strukturu

- trunk - hlavní větev
- tag - ukládání štítků řešení - obraz aktuálního stavu řešení v daném okamžiku
- branch - vývojové větve

7 .4.3.5 Vylepšení

Během vývoje se postupně objevovaly problémy s nastavením některých aspektů Continuous Integration popsanými výše. Nejviditelnějším problémem byla definice nových úloh na integračním serveru hlavně ve fázích prototypování, kdy dochází k časté změně úloh na serveru. V základním nastavení bylo nutné vytvořit novou úlohu pomocí konfiguračního souboru, server zastavit, upravit veškeré konfigurační soubory a následně server opět spustit. Tento proces, ačkoliv konfigurace úloh nezabrala mnoho času díky sdíleným souborům s nastavením, plně nevyhovoval myšlence Continuous Integration a také jistou měrou omezoval ostatní úlohy častým vypínáním serveru. K řešení jsem využil integračního serveru tak, aby sám sebe konfiguroval v případě změn. Tohoto bylo dosaženo umístěním veškeré konfigurace serveru do speciálního projektu v SVN a vytvoření dedikované úlohy právě na integračním serveru, který tento repositář monitoroval pro změny v konfiguracích. V případě změny v jakékoliv konfiguraci úlohy nebo obecných nastavení došlo k aktualizaci těchto konfigurací na serveru a restartování služby serveru. Takto byl zkrácen čas na změny nastavení nebo přidání nových úloh na několik vteřin a proces byl plně automatizován [1]. Konfigurace serveru CruiseControl.NET a další náležitosti jsou součástí přílohy A .3.

Postupem času s dalšími projekty docházelo k rozšiřování NAnt skriptů o další podpůrné funkcionality. Správa těchto rozšíření se stala velmi náročnou a začalo se objevovat množství chyb způsobených opomenutou změnou v nastavení NAnt různých projektů. Pro odstranění tohoto problému bylo opět využito integračního serveru a serveru SVN. Veškerá rozšíření NAnt byla generalizována tak, aby byla nezávislá na projektech a byla všechna umístěna na server SVN. Na integračním serveru, opět jako v případě konfigurací vlastního serveru, vznikla úloha, která toto umístění monitorovala o změny a aktualizovala změněné soubory NAnt skriptů na integračním serveru. Konfigurace úloh na serveru pak byla rozšířena o aktualizaci těchto skriptů před každým sestavením řešení. Toto opět

zrychlilo změny ve skriptech a odstranilo nežádoucí chyby způsobené opomenutím některé konfigurace pro konkrétní projekt. Skripty a konfigurace úloh jsou součástí přílohy A .3.

Dalším vylepšením, pramenícím z hledání úspory času pro sestavení [1], byla integrace kontroly pravidel StyleCop [3] již v průběhu kompilování řešení. V původním nastavení, byl StyleCop využit jako externí program a kontrola zdrojových kódů probíhala až po sestavení aplikace, což prodlužovalo čas potřebný k sestavení. Integrace StyleCop přímo do sestavení aplikace nejen, že urychlila vlastní sestavení řešení, ale umožnila zavést tuto kontrolu již při sestavení na stroji vývojáře a tak odhalila porušení pravidel již lokálně a nebylo nutné se kvůli opravě chyb vracet až po sestavení na integračním stroji. Kontrola těchto pravidel byla nastavena nejdříve pouze pro nové soubory a projekty přidané do řešení a postupně byla rozšiřována o již stávající soubory řešení. Skripty, které slouží ke konfiguraci projektů a řešení pro kontrolu StyleCop pomocí tohoto postupného systému jsou součástí přílohy A .3 této práce.

7 .5 Uzavření projektu

Poslední fází procesního řízení vývoje software, je předání řešení a uzavření projektu.

Pro podporu předání a nasazení software je použito powershell skriptu, který obstarává úkony nutné k nasazení software. Tento skript je opět plně nezávislý na konkrétním řešení a využívá obecné konfigurace nasazení aplikace ve které jsou specifikovány různé aspekty řešení, jako je umístění databázového serveru a přístupy, služby aplikace, umístění záloh aplikace a databáze, složka instalace aplikace a další. Tento skript načte zmíněnou konfiguraci a provede veškeré nutné aktivity pro nasazení aplikace. Při předchozím nasazení bez použití tohoto skriptu, bylo nutno spouštět několik příkazů a byla nutná kontrola několika členů vývojového týmu, protože docházelo k chybám. Dalším problémem byl čas nutný k nasazení, který dosahoval řádově desítek minut. Využitím tohoto skriptu je možné aplikaci nasadit během 15 minut bez dalších aktivit členů týmu nebo dalších kontrol. Skript se využívá, krom prvotního nasazení software zákazníkovi, hlavně v rámci servisní činnosti a záručních oprav.

Po úspěšném předání řešení nastávají aktivity vedoucí k finálnímu ukončení projektu. V této fázi je uzavřen projekt v systému JIRA a pomocí interního nástroje ELVAC SOLUTIONS jsou zhodnoceny projektové KPI jako jsou očekávaný a reálný zisk, očekávané a reálně odpracované hodiny, hodiny nutné k řízení projektu (odpracované hodiny, které neslouží k vývoji software).

Veškeré takto získané KPI jsou pak prezentovány v rámci retrospektivy projektu. Retrospektiva projektu vyhází z metodiky SCRUM a stejně jako na denních setkání členů vývojového týmu se zde prezentují ověřené praktiky (ve kterých je dobré pokračovat), problémové věci a návrhy na zlepšení. Toto jednaní může být za přítomnosti zákazníka (v tom případě ovšem neprezentujeme finanční výsledky, ale pouze hodinové výsledky), který nám takto dává zpětnou vazbu na naši práci a mnohdy je cenným zdrojem informací a návrhů na procesní vylepšení vývoje.

K dokumentaci retrospektivy je opět využito projektového prostoru v systému Confluence a šablony pro retrospektivu projektu.

Po ukončení projektu je v systému JIRA uložen celkový obraz průběhu projektu z pohledu nákladů, času, jednotlivých úloh, milníků, testů, chyb, oprav v release a dalších náležitostí souvisejícího s vývojem software a podkladů pro hodnocení projektů. V systému Confluence je uložena veškerá dokumentace projektu - zápisy z jednání, záznamy změnového řízení, záznamy retrospektivy projektu, požadavky zákazníka a úkoly z nich plynoucí s propojením do systému JIRA a další informace. V systému Confluence lze tedy nalézt veškeré náležitosti projektového řízení s jednoznačnou vazbou a kontrolou na vykonané úkoly. V SVN repositáři řešení jsou pak uloženy zdrojové kódy řešení a to i různé verze vytvořené v rámci jednotlivých milníků pomocí štítků vytvořených integračním serverem.

Tento způsob práce plně podporuje procesy a nutné aktivity z pohledu normy ISO 12207 [10]. V rámci certifikace v roce 2014 tento systém práce byl [10] shledán vyhovujícím a podporujícím veškeré procesy této normy.

8 Implementace Continuous Integration v prostředí JAVA

Pro nastavení Continuous Integration procesů pro vývoj v prostředí JAVA lze využít veškerých popsaných aktivit, KPI, procesů a dokumentace popsaných v předchozích kapitolách.

Projektové řízení a jeho fáze jsou nezávislé na konkrétní technologii vývoje software. Obecně lze i říci, že organizace projektů a procesy projektového řízení zde uvedené lze využít i pro jiné projekty než je vývoj software. Nástroje JIRA a Confluence, zvolené ve společnosti ELVAC SOLUTIONS, pro řízení projektů lze použít obecně. JIRA v sobě kombinuje nástroj pro plánování, řízení prací a monitorování postupu a nákladů a Confluence slouží jako správa dokumentů a podpora organizačních náležitostí projektu.

Nástroje pro vlastní vývoj už je vhodné upravit podle technologie pod kterou software vyvíjíme. I když například integrační server CruiseControl.NET plně vychází z platformy .NET, jeho použití pro vývoj v jazyce JAVA je také možné. Obecným doporučením je ale využít nástroje, který se více hodí k dané technologii a nějakým způsobem ji rozšiřuje nebo alespoň nativně podporuje a využívá.

Pro vývoj v prostředí JAVA bych tedy doporučil jako integrační server Jenkins (více v kapitole 5 .3). Tento server nativně podporuje jazyk JAVA, má velkou uživatelskou základnu a je stále rozšiřován a doplňován o nové funkcionality. Úlohy na tomto serveru mohou zůstat stejné jako v případě vývoje .NET (continuous integration úloha, dokumentace, nasazení software), protože jde o obecné aktivity každého vývoje software. Při vývoji v .NET byl pro integrační skripty použit nástroj NAnt. Jak už je z jeho jména vidět je to v podstatě klon nástroje Ant převedený do prostředí .NET. Pro definici integračních skriptů bych tedy použil tohoto nástroje, jenž je vytvořen přímo v jazyce JAVA a nativně podporuje veškeré aktivity nutné nejen k sestavení aplikace. Jako úložiště zdrojových kódů bych zvolil nástroj Git, jenž je velmi hojně využíván open source komunitou a jeho podpora je na velmi vysoké úrovni. Pro zajištění kvality, testování řešení a dalších podpůrných aktivit bych zvolil nástroje vycházející z platformy JAVA. Pro testování přichází v úvahu JUnit a pro kontrolu kvality kódu a dodržování standardů nástroj CheckStyle.

Stejně jako i v případě implementace Continuous Integration pro projekty používající platformu .NET je implementace podpory těchto procesů na JAVA platformě nenákladná z pohledu investovaných prostředků do nástrojů.

9 Závěr

Jako Continuous Integration Manager ve společnosti ELVAC SOLUTIONS bylo mým primárním úkolem zlepšení kvality vývoje softwarových produktů a kvality vyvíjených produktů společnosti, automatizace běžných činností ve společnosti a zajištění podpory vývojových týmů při těchto aktivitách. Druhým cílem bylo definovat a nastavit procesní rámec projektového řízení ve společnosti, nástroje pro jeho podporu, kontrolní body a monitorování projektů pomocí definice vhodných výkonnostních ukazatelů s ohledem na certifikaci normy ISO 9001 a 12207.

Jako první jsem definoval a nastavil procesy projektového řízení ve společnosti. Jako nástroje podporující nově definované procesy projektového řízení jsem zvolil nástroje JIRA a Confluence, které již společnost okrajově využívala. Ze své pozice jsem se stal správcem těchto nástrojů a nastavil jsem je tak, aby plně podporovaly nově definované procesy pro řízení a kontrolu projektů.

Současně se zaváděním nových procesů projektového řízení, jsem připravil instanci integračního serveru CruiseControl.NET. V první řadě jsem pomocí zde prezentovaného nástroje NAnt připravil kompletní soubor skriptů, které odpovídaly za sestavení aplikace a její nasazení do interních testovacích prostředí provozovaných v rámci ELVAC SOLUTIONS. Následně jsem implementoval rozšíření sestavení o automatické testy řešení, kontrolu kvalitativních metrik, publikování dokumentace řešení a další aktivity nezbytné pro zajištění kompletní podpory vývoje softwarových produktů. Posledním krokem v rámci životního cyklu vývoje software bylo vytvoření powershell skriptů pro nasazení řešení v produkčních prostředích našich zákazníků.

Veškerá nastavení a procesy vytvořené během mé práce ve společnosti, jsou nyní denně využívány při vývoji software, řízení projektů a sledování výkonnostních ukazatelů (KPI). V počátcích definice a implementace bylo těžké přesvědčit zaměstnance společnosti o výhodách nových postupů, které představovaly zásah do zažitých postupů a vyžadovaly jistou součinnost i z jejich strany. Jejich postupným zaváděním byly ale demonstrovány výhody nových procesů při jejich každodenní práci. Nakonec se jako hlavní výhoda pro zaměstnance ukázala úspora času zaměstnanců strávené na vývoji a ověření správnosti zadaných úkolů a situace se začala rychle zlepšovat.

Integrační server je v současnosti standardní součástí pracovních postupů v ELVAC SOLUTIONS a stal se mezi zaměstnanci základním ukazatelem kvality odvedené práce. Došlo k odklonu od věty "U mě to funguje" k větě "Jde to sestavit na integračním severu". Také se mezi zaměstnanci společnosti přirozeně vytvořil pocit odpovědnosti za stav integrační úlohy na serveru, kdy každý z členů vývojového týmu se snaží případné chyby ihned opravit, tak aby ostatní členové týmu mohli nerušeně pokračovat ve vývoji nad řešením, které je funkční a otestované. Automatizace veškerých běžných a opakujících se činností umožnila zaměstnancům společnosti se věnovat nejen vývoji, ale také i studiu nových technologií a rozvoji stávajících produktů a řešení.

Podpora projektového řízení vytvořila návyky orientované na kvalitu a organizaci projektů s ohledem na nutné artefakty projektového řízení (záznamy z jednání, definice změnového řízení, analýza rizik a podobné), kdy výsledkem je jasná viditelnost a přehlednost procesů. Projekty lze jednodušeji porovnávat z pohledu hlavně finančních výsledků pro společnost, ale také nastavené procesy umožnily poučit se z chyb a problémů ostatních projektů.

Implementací Continuous Integration ve společnosti ELVAC SOLUTIONS jsem dosáhl mnoha vylepšení, jak procesních tak kvalitativních. Z těchto vylepšení je nejviditelnější časová úspora při vývoji řešení (automatické testování, automatizace běžných činností, rychlá zpětná vazba, kontrola kvality zdrojových kódů a další). Dalším velkým vylepšením je nynější podpora procesů projektového řízení a jejich integrace do vývojového procesu, kdy jsou zaměstnanci obeznámeni s řízením projektů jako celku a cítí tak vetší odpovědnost za projekt jako celek a snaží se odvádět co nejlepší práci.

Procesní řízení pomocí metodiky SCRUM, které jsem zavedl, nastavilo pravidla a prostředí pro častou komunikaci se zákazníkem. Jeho účast v projektu položila základ k transformaci k novému způsobu práce, kdy zákazník je centrálním bodem při vývoji. Toto je podporováno zpřístupněním systému JIRA pro zákazníky společnosti a umožňuje jim tak lepší vhled do postupu prací, možnost definovat funkcionalitu dodávaného software a aktivně a pravidelně se podílet na vývoji na základě jeho aktuálních potřeb.

Stávající implementace Continuous Integration procesů stále není finální a stále dochází k postupnému zlepšování směrem k co největší nezávislosti a automatizaci. Dosavadní procesy, implementované v rámci této práce, přispěly k bezproblémové certifikaci ISO 12207 [10] - standardu zabývajícího se právě vývojem software a certifikaci ISO 9001 [11] zabývající se řízením dokumentace a procesů ve firmě. Časová úspora na straně vývoje pak znamená nemalou úsporu finančních prostředků. Zaměstnanci a členové vývojových týmů mají pozitivní přístup k novému způsobu práce a neustále přicházejí s nápady, jak ještě více procesy a běžné činnosti automatizovat. Vedení společnosti oceňuje hlavně procesní stránku Continuous Integration, která podporuje sledování výkonnostních ukazatelů projektů, jenž usnadňují sledování finančních toků ve společnosti a zlepšují nakládání s investicemi.

10 Literatura

[1] DUVALL Paul M., MATYAS Steve, GLOVER Andrew *Continuous Integration: Improving Software Quality and Reducing Risk.* Vydáno 2007, 1. edice, Addison-Wesley Professional, 336s., ISBN-13: 978-0-321-33638-5

[2] HUMBLE Jez, FARLEY David *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.* Vydáno 2010, 1. edice, Addison-Wesley Professional, 512s., ISBN-13: 978-0-321-60191-9

[3] KAWALEROWICZ Marcin *Continuous Integration in .NET.* Vydáno 2011, 1. edice, Manning Publications, 375s., ISBN-13: 978-1935182559

[4] *Martin Fowler - Blogs on Continuous Integration* `http://martinfowler.com/`

[5] *Manifest vývoje software použitím agilních metodik* `http://agilemanifesto.org/`

[6] *Stránky produktu JIRA firmy Atlassian* `https://www.atlassian.com/software/jira`

[7] *Stránky produktu Confluence firmy Atlassian* `https://www.atlassian.com/software/confluence`

[8] *Internetová prezentace nástroje CruiseControl.NET* `http://www.cruisecontrolnet.org/`

[9] *Stránky nástroje NAnt* `http://nant.sourceforge.net/`

[10] *ISO/IEC 12207:2008 Systems and software engineering – Software life cycle processes*

[11] *ISO 9001:2008 Quality management systems – Requirements*

A Přílohy

A .1 Konfigurace úlohy na integračním serveru

```
<cb:config-template xmlns:cb="urn:ccnet.config.builder">
  <cb:scope ProjectName="solution" ProjectType="_RELEASE">
    <project name="$(ProjectName)$(ProjectType)" queue="solution" queuePriority="1">
      <webURL>
        <a href="http://srv-solbuild2:8080/ccnet/server/local/project/$(ProjectName)$(
            ProjectType)/ViewLatestBuildReport.aspx">http://srv-solbuild2:8080/ccnet/server/
            local/project/$(ProjectName)$(ProjectType)/ViewLatestBuildReport.aspx</a>
      </webURL>
      <modificationDelaySeconds>10</modificationDelaySeconds>
      <workingDirectory>$(WorkingDir)$(ProjectName)$(ProjectType)</workingDirectory>
      <artifactDirectory>$(MainArtifactsDir)$(ProjectName)$(ProjectType)$(ArtifactsDir)</
          artifactDirectory>
      <category>solution</category>
      <sourcecontrol type="svn">
        <cb:svn_ci />
        <trunkUrl>$(SVNServer)$(ProjectName)/trunk</trunkUrl>
        <workingDirectory>$(WorkingDir)$(ProjectName)$(ProjectType)</workingDirectory>
      </sourcecontrol>
      <tasks>
        <exec>
          <executable>xcopy.exe</executable>
          <buildArgs>/y $(WorkingDir)NAnt_configuration\*.build $(WorkingDir)$(
              ProjectName)$(ProjectType)\</buildArgs>
          <buildTimeoutSeconds>60</buildTimeoutSeconds>
        </exec>
        <nant>
          <cb:nant_ci />
          <baseDirectory>$(WorkingDir)$(ProjectName)$(ProjectType)</baseDirectory>
          <targetList>
            <target>$[ReleaseType|full_release]</target>
          </targetList>
          <buildTimeoutSeconds>6000</buildTimeoutSeconds>
          <buildArgs>-D:ProjectName.CI="$(ProjectName)$(ProjectType)" -D:SvnName="$(
              ProjectName)" -D:build.type="$[BuildType|Release]"</buildArgs>
        </nant>
        <nant>
          <cb:nant_ci />
          <baseDirectory>$(WorkingDir)$(ProjectName)$(ProjectType)</baseDirectory>
          <targetList>
            <target>$[CreateTag|clean]</target>
          </targetList>
          <buildTimeoutSeconds>6000</buildTimeoutSeconds>
          <buildArgs>-D:ProjectName.CI="$(ProjectName)$(ProjectType)" -D:SvnName="$(
              ProjectName)" -D:build.type="$[BuildType|Release]"</buildArgs>
        </nant>
      </tasks>
      <publishers>
        <merge>
          <files>
```

```
                    <cb:merge />
                </files>
            </merge>
            <cb:common_publishers />
            <cb:revert_solution  />
        </publishers>
        <parameters>
            <cb:release_parameters />
        </parameters>
        <labeller  type="sharedLabeller">
            <sharedLabelFilePath>$(SharedLabelsDir)solution_shared_label.txt</
                sharedLabelFilePath>
            <incrementOnFailure>true</incrementOnFailure>
        </labeller>
        <!--<labeller type="defaultlabeller">
            <initialBuildLabel>5005</initialBuildLabel>
            <incrementOnFailure>true</incrementOnFailure>
        </labeller>-->
        <security type="defaultProjectSecurity"  defaultRight="Deny" guest="Guest">
            <permissions>
                <cb:security_ci  />
            </permissions>
        </security>
    </project>
</cb:scope>
</cb:config-template>
```

Výpis 1: Konfigurace úlohy na integračním serveru

A .2 Konfigurace sestavení pomocí NAnt

```
<?xml version="1.0" ?>
<project name="solution" default="help" basedir=".">
  <description>Build script.</description>
  <!-- GENERAL CCNET PROJECT PROPERTIES -->

  <!-- CCNET job name -->
  <property name="verbose" value="true" unless="${property::exists('verbose')}"/>
  <!--Projec name on CruiseControl.NET-->
  <property name="ProjectName.CI" unless="${property::exists('ProjectName.CI')}" value="
      solution" overwrite="true"/>
  <!-- Project artifacts directory (where all the results are outputted) -->
  <property name="Artifacts.dir" value="c:\CCNetSandBox\${ProjectName.CI}" verbose="${
      verbose}"/>
  <property name="Artifacts.dir" value="${environment::get-variable('CCNetArtifactDirectory')}"
      if="${property::exists('CCNetArtifactDirectory')}"/>
  <!-- Project working directory -->
  <property name="Working.dir" value="${project::get-base-directory()}" verbose="${verbose}"/
      >
  <property name="Working.dir" value="${environment::get-variable('CCNetWorkingDirectory')}"
      if="${property::exists('CCNetWorkingDirectory')}" />
  <!-- Project name in SVN -->
  <property name="SvnName" unless="${property::exists('SvnName')}" value="solution"/>
  <property name="SVNServer" unless="${property::exists('SVNServer')}" value="https://srv-
      soldev:8443/svn/"/>
  <property name="SvnURI" value="${SVNServer}${SvnName}/trunk" overwrite="true" />

  <!-- If multiple solutions to build please separate by ';' -->
  <!-- e.g. Code\EDF_Build\EDF_Build.sln;Code\EDF_Build\unknow.csproj -->
  <property name="SolutionPath" value="Code\Apps.solution.Web.sln;Code\Apps.solution.sln;
      Code\Tools.DatabaseUpdater.sln;Code\Model.sln"/>
  <!--Leave empty if not needed-->
  <property name="TaskServer.sln" value="Code\TaskServer.sln"/>
  <!--<property name="separate.solutions" value="false" readonly="true"/>-->
  <!-- .dll containing tests -->
  <!--<property name="TestContainer.file" value=""/> -->
  <!--Connection string to test database-->
  <property name="TestDatabaseConnectionString" value="XpoProvider=MSSqlServer;Data
      Source=SRV-SOLSQLDEV;Initial Catalog=solution_CI;Persist Security Info=True;User ID
      =sa;Password=password"/>
  <!-- Assembly info file location -->
  <property name="AssemblyInfo.location" value="${Working.dir}\Code\Apps\solution\XAF.Win\
      Properties\AssemblyInfo.cs"/>
  <!--Configuration file for doxygen documentation-->
  <property name="DoxyGen.config.file" value="${Working.dir}\Docs\${SvnName}.config"/>
  <!--<property name="stylecop.config.file" value="solution.StyleCopCheck.proj"/>-->

  <property name="ReleaseOutput.setup.file" value="" />
  <property name="chm.out.file" value="${SvnName}.chm" />

  <!-- MSBuild version used to build this solution -->
```

```
<property name="NET.version" value="v4.0.30319" unless="${property::exists('.NET.version')}"/
    >
<!-- Target framework parameter /tv for MSBuidl -->
<property name="tools.version" value="/tv:4.0" readonly="true"/>
<!-- DatabaseUpdater tool properties -->
<property name="DatabaseUpdaterFileName" value="ELVAC.solution.Tools.
    DatabaseUpdaterConsole.exe"/>
<property name="DatabaseUpdaterParameters" value="-T1 -DROPDATABASE -
    NOWAITONEND" unless="${property::exists('DatabaseUpdaterParameters')}"/>


<!-- DIRECTORIES -->
<!--Here you can specify the targetdirectories per release type - FTP, customer, local test
    release, etc.-->
<property name="Documentation.target.dir" value="${Artifacts.dir}\Docs" />
<property name="References.dir" value="${Working.dir}\References" />
<property name="ReleaseOutput.temp.dir" value="C:\Temp\${ProjectName.CI}"/>
<property name="ReleaseOutput.temp.zip.dir" value="${ReleaseOutput.temp.dir}_ZIP"/>
<property name="ReleaseOutput.dir" value="\\srv-solsql\DATA\XAF_UPDATES\${SvnName}"
    />
<property name="ReleaseOutput.ftp.dir" value="\\srv-web2\FTP_Solutions\Outbound\${
    SvnName}" />
<property name="ReleaseOutput.web.dir" value="\\srv-solsqldev\wwwroot\solution" />
<property name="BuildOutput.dir" value="${Artifacts.dir}\out"/>
<property name="TaskServerOutput.dir" value="${Artifacts.dir}\TaskServer"/>
<property name="ApplicationConfig.location" value="${BuildOutput.dir}\solution.exe.config"/>

<!-- VERSION FOR RELEASE -->
<property name="build.major" value="1"/>
<property name="build.minor" value="0"/>
<property name="build.build" value="0"/>
<property name="build.build" value= "${environment::get-variable('CCNetNumericLabel')}" if="
    ${property::exists('CCNetNumericLabel')}"/>
<property name="svn.revision" value="0"/>
<property name="counter" value="0"/>

<!-- TASK SERVER SERVICE -->
<!-- Specify the machine name in UNC ( \\srvname ) -->
<!--Uncomment if location should be different from srv-solxaftest-->
<!--<property name="TaskServerService.machine" unless="${property::exists('
    TaskServerService.machine')}" value="\\srv-solxaftest"/>-->
<!--Service will be installed to c:\Program Files\ELVA SOLUTIONS directory-->
<property name="TaskServerService.name" value="ELVAC.solution.TaskServer.Service"/>
<property name="TaskServerService.binary" value="ELVAC.solution.TaskServer.Service.exe"/>

<!-- TESTING -->
<property name="dll.to.analyze" value="XAF.Module.Win.dll;XAF.Module.dll;ELVAC.solution.
    Model.dll"/>

<!--LICENSING-->
<property name="license.file.to.generate" value="${Working.dir}\Code\Apps\solution\XAF.Win
    \solution_license.cs" />
<property name="license.namespace" value="ELVAC.solution" />
<property name="license.classname" value="Licensing" />
```

```
<!--DOCUMENTATION-->
<property name="CFL.doc.url" value='http://confluence.elvacsolutions.eu/rest/scroll-office/1.0/
    sync-export?exportSchemeId=TECHIS030-0A00C26F0145405D0EC7865700FE9191"&
    amp;"rootPageId=106758232"&amp;"os_authType=basic'/>
<property name="CFL.doc.target" value="${Artifacts.dir}"/>
<property name="CFL.doc.name" value="${SvnName}.docx"/>

<!-- Release target for testing purposes -->
<target name="test_release">
  <property name="test_release" value="true" />
  <if test="${build.type=='Release'}">
    <call target="clean_release_folder"/>
  </if>
  <property name="compress.files" value="true" unless="${property::exists('compress.files')}"/
      >
  <!--WEB release has to be first !!!! -->
  <property name="ReleaseOutput.web.dir" value="${ReleaseOutput.temp.dir}_WEB"/>
  <call target="release_web_application"/>
  <!-- Release locally or to update server !!!! -->
  <property name="ReleaseOutput.dir" value="${ReleaseOutput.temp.dir}"/>
  <call target="release_application"/>
  <call target="ftp_upload_release" if="${compress.files}"/>
</target>

<!-- Release target -->
<target name="release">
  <if test="${build.type=='Release'}">
    <call target="clean_release_folder"/>
  </if>
  <property name="compress.files" value="true" unless="${property::exists('compress.files')}"/
      >
  <!--WEB release has to be first !!!! -->
  <call target="release_web_application"/>
  <!-- Release locally or to update server !!!! -->
  <call target="release_application"/>
  <call target="ftp_upload_release" if="${compress.files}"/>
</target>

</project>
```

Výpis 2: Nastavení sestavení aplikace

A .3 Příloha na CD/DVD

Následující seznam souborů je součástí přílohy této diplomové práce. Tento seznam obsahuje pouze důležité soubory a vlastní příloha obsahuje doplňující podpůrné nástroje. Citlivé údaje jako jsou hesla, jména databází, přihlašovací údaje k interním systémům ELVAC SOLUTIONS a další byly z těchto souborů odstraněny.

Podpůrné nástroje třetích stran nejsou distribuovány jako součást této práce, ale je zde uveden odkaz ke stažení daného nástroje z oficiálních zdrojů.

Adresář	**Soubor**	**Komentář**
CruiseControl.NET	build_general_configuration.xml	Obecná konfigurace CruiseControl.NET, která je sdílena mezí úlohami integračního serveru.
CruiseControl.NET	ccnet.config	Konfigurace úloh, které jsou vykonávány na integračním serveru.
CruiseControl.NET	ccnet.exe.config	Konfigurační soubor integračního serveru.
CruiseControl.NET	CCNET_configuration.xml	Úloha integračního serveru zajišťující automatickou aktualizaci integračního serveru při změně.
CruiseControl.NET	ccservice.exe.config	Konfigurace služby integračního serveru. Zde je konfigurováno umístění při jejichž změně se server má restartovat.

CruiseControl.NET	NAnt_configuration.xml	Úloha integračního serveru zajišťující automatickou aktualizaci skriptů NAnt.
BuildServerJobDefinitions	solution_CI.xml	Definice integrační úlohy pro pravidelnou kontrolu řešení. V rámci této úlohy je řešení sestaveno, otestováno, proběhne kontrola kvalitativnívh metrik a další.
BuildServerJobDefinitions	solution_documentation.xml	Definice integrační úlohy pro generování dokumentace řešení.
BuildServerJobDefinitions	solution_RELEASE.xml	Definice integrační úlohy pro různé typy nasazení software - testovací, produkční, FTP a další.
BuildServerJobDefinitions	solution_INSTALLER_RELEASE.xml	Definice integrační úlohy pro vytvoření instalátoru řešení.
DeploymentScripts	!ccnet_db_update.bat	Volání powershell skriptu pro potřeby nasazení řešení pomocí integračního serveru.

DeploymentScripts	!deploy_me.bat	Volání powershell skriptu pro naszení řešení u zákazníka - různé konfigurace.
DeploymentScripts	!deploy_me_change_config.bat	Volání powershell skriptu pro naszení řešení u zákazníka - různé konfigurace.
DeploymentScripts	!deploy_me_configure_taskserver.bat	Volání powershell skriptu pro naszení řešení u zákazníka - různé konfigurace.
DeploymentScripts	!deploy_me_dont_update_version_server.bat	Volání powershell skriptu pro naszení řešení u zákazníka - různé konfigurace.
DeploymentScripts	!deploy_me_recreate_db.bat	Volání powershell skriptu pro naszení řešení u zákazníka - různé konfigurace.
DeploymentScripts	!deploy_me_update_model.bat	Volání powershell skriptu pro naszení řešení u zákazníka - různé konfigurace.
DeploymentScripts	!deploy_web.bat	Volání powershell skriptu pro naszení řešení u zákazníka - různé konfigurace.
DeploymentScripts	!deploy_web_change_config.bat	Volání powershell skriptu pro naszení řešení u zákazníka - různé konfigurace.

DeploymentScripts	!sql_and_app_backup.bat	Volání powershell skriptu pro datbázové operace při nasazení řešení.
DeploymentScripts	!sql_backup.bat	Volání powershell skriptu pro datbázové operace při nasazení řešení.
DeploymentScripts	!sql_backup_stop_and_start_services.bat	Volání powershell skriptu pro datbázové operace při nasazení řešení.
DeploymentScripts	!sql_backup_stop_services.bat	Volání powershell skriptu pro datbázové operace při nasazení řešení.
DeploymentScripts	!sql_backup_to_ftp.bat	Volání powershell skriptu pro datbázové operace při nasazení řešení.
DeploymentScripts	!sql_restore_latest_bak.bat	Volání powershell skriptu pro datbázové operace při nasazení řešení.
DeploymentScripts	!sql_restore_latest_bak_and_start_service.bat	Volání powershell skriptu pro datbázové operace při nasazení řešení.
DeploymentScripts	!sql_restore_truncate_versions.bat	Volání powershell skriptu pro datbázové operace při nasazení řešení.

DeploymentScripts	!upload_latest_backup_to_ftp.bat	Volání powershell skriptu pro datbázové operace při nasazení řešení.
DeploymentScripts	ccnet_deployment.config	Konfigurace nasazení řešení pomocí integračního serveru.
DeploymentScripts	DeployApplication.ps1	Powershell skript pro nasazení řešení. Umožňuje konfigurace, služeb, databáze, úložiště řešení a další.
DeploymentScripts	deployment.config	Konfigurace nasazení řešení u zákazníka.
DeploymentScripts	DeployWeb.ps1	Powershell skript pro nasazení webu řešení. Umožňuje konfigurace, služeb, databáze, úložiště řešení a další.
DeploymentScripts	SQL_db_backup.ps1	Powershell skript pro operace při zálohování databáze řešení.
DeploymentScripts	SQL_db_restore.ps1	Powershell skript pro operace při obnově databáze řešení.
DeploymentScripts	SQL_db_update.ps1	Powershell skript pro operace při aktualizaci databáze řešení.
NAnt	build.build	Obecná konfigurace Nant pro sestavení řešení na integračním serveru,

NAnt	buildInstaller.build	Konfigurace sestavení a publikování instalátorů řešení.
NAnt	compile.build	NAnt úlohy související se sestavením řešeí.
NAnt	default.build	Obecné konfigurace NAnt a rozcestník pro volání dalších úloh NAnt.
NAnt	documentation.build	NAnt úlohy související s publikováním dokumentace řešení.
NAnt	edf_release.build	NAnt úlohy související s publikováním release interního nástroje ELVAC SOLUTIONS.
NAnt	helpers.build	Pomocné úlohy pro NAnt.
NAnt	qa.build	NAnt úlohy pro kontrolu kvality řešení.
NAnt	release.build	NAnt úlohy pro vydání řešení
NAnt	settings.build	Konfigurace NAnt úloh.
ToolBox	DownloadCFLDocumentation.ps1	Powershell skript integračního serveru pro stažení dokumentace řešení publikované v systému Confluence.
ToolBox	GetAssemblyVersions.ps1	Získání verzí knihoven vydané aplikace.

ToolBox	query\service.bat	Zjistí status služby TaskServer pro potřeby nasazení software.
ToolBox	release\notes.ps1	Zajišťuje generování release notes v xml ormátu pro integrační server.
ToolBox	svn\info.bat	Dotaz zda existuje v daném repositáři adresář se stejným jménem jako parametr.
ToolBox	ThoughtWorks.CruiseControl.MSBuild.dll	Knihovna pro podporu logování výstupu MSBuild na integračním severu.
ToolBox	Word2PDF.ps1	Převod Word dokumentu na PDF pomocí powershell. Využívá se na integračním serveru v rámci publikování dokumentace.
ToolBox\doxygen		Nástroj DoxyGen pro generování dokumentace. `www.doxygen.org`
ToolBox\iconv		Nástroj pro převod řešení do UTF-8 Nástroj na serveru SourceForge.net

ToolBox\MSCodeCoverageToXML		Převod výtupů pokrytí kódu unit testy pro potřeby prezentace na integračním serveru.
ToolBox\nant-0.92		Nástroj NAnt. `http://nant.sourceforge.net/`
ToolBox\nantcontrib-0.92		Rozšiřující knihovny NAnt. `http://nantcontrib.sourceforge.net/`
DP		Zdrojové kódy diplomové práce v LaTeX.
JIRA	ELVAC-Project-Development-Tasks-Workflow.jwb	Pracovní postup definovaný v systému JIRA pro podporu vývoje projektů.
PM	Direktiva projektu.docx	Šablona pro direktivu projektu.
PM	Projektový plán.docx	Šablona pro projektový plán.
Powershell		Různé podpůrné powershell skripty, které jsou využity v celém procesu.
VisualSVN\hooks		Nástorje na kontrolu kódování řešení do UTF-8 na straně SVN serveru.

Table 1: Seznam příloh diplomové práce

A .4 Zjednodušený popis nastavení serveru a konfigurace úloh

Tato část slouží jako nástin konfigurace integračního serveru a úloh zajišťující integraci řešení do celku v rámci serveru. Vzhledem k tomu, že tato práce byla implementována pro konkrétní společnost používající specifické, placené nástroje, je tento popis spíše příkladem aktivit nutných k definici Continuous Integration. Součástí přílohy jsou reálné konfigurace integračních úloh, které ale bez kontextu dané společnosti nebudou pracovat správně. Pokusím se zde tedy nastínit úpravu těchto skriptů (nutné parametry) pro jiné řešení nad platformou .NET. Tyto úpravy vyžadují zkušenosti s oblastí Continuous Integration, vývojem v .NET a obecně nejsou nijak triviální. Tato konfigurace je aplikovatelná pouze pro operační systém Windows.

Prerekvizity

- Adresář ToolBox je umístěn na disku C počítače (v případě jiného umístění je nutno upravit konfigurace sestavení)
- Spustitelné soubory nástrojů NAnt, svn, doxygen a iconv musí být definované v proměnné prostředí PATH, tak aby je bylo možno volat v jakémkoliv kontextu.
- Nainstalován MSBuild, jenž je součástí Microsoft .NET framework verze 4.0
- Nainstalován nástroj FxCop a StyleCop
- Nainstalován server SVN
- Na serveru SVN jsou vytvořeny repositáře pro konfiguraci NAnt (NAnt_configuration) a CruiseControl.NET (CCNET_configuration) a v nich jsou umístěny odpovídající soubory jenž jsou součástí přílohy A .3
- Nainstalována služba IIS pro běh serveru CruiseControl.NET
- Vytvořen adresář CCNetSandBox na disku C (v případě jiného umístění je nutno upravit konfigurace sestavení)

Pro ověření správného nastavení lze v adresáři NAnt, který je součástí přílohy práce, spustit příkaz nant. V případě správné konfigurace se zobrazí nápověda daného projektu. V tuto chvíli je možno vykonávat příkazy NAnt pro sestavení, vydání, generování dokumentace. Je však nutné připravit konfiguraci sestavení dle aktuálního projektu.

Pro kontrolu instalace CruiseControl.NET a nastavení integračního serveru a IIS stačí navštívit stránky webového rozhraní serveru.

Konfigurace integračního serveru

Konfigurace integračního serveru je popsána na stránkách produktu CruiseControl.NET [8], kde lze nalézt i instalátor integračního serveru. Po instalaci je nutné nahradit soubory

ccnet.config, **ccnet.exe.config** a **ccservice.config** soubory, které jsou součástí přílohy této práce. Zde je nutné dodržet umístění ostatních souborů, tak jak jsou definovány v rámci konfiguračních souborů, nebo upravit odpovídající položky konfigurace.

V souboru **build_general_configuration.xml** je nutné nastavit přihlášení a heslo k SCM SVN, konfigurace SMTP pro email notifikaci a případně upravit cesty k adresářům do kterých si integrační server ukládá aktuální verzi řešení, výsledky sestavení a další nastavení.

- WorkingDir - pracovní adresář, kde jsou uloženy řešení k sestavení
- MainArtifactsDir - adresář k uložení výsledků sestavení - artefaktů
- SVNServer - adresa serveru SVN

Po tomto nastavení by na webovém rozhraní integračního serveru měly být zobrazeny tři úlohy CCNET_configuration, NAnt_configuration a ES_ProtoExpress_CI. Úlohy by se měly automaticky spustit, ale pokud nejsou správně nakonfigurovány, tak skončí chybou, která bude prezentována email notifikací uživateli (pokud byl tento správně nastaven).

Konfigurace úlohy integračního serveru

V konfiguraci úloh (viz. příklad A .1) je nutné specifikovat adresu SVN serveru a repositáře, která obsahuje zdrojová kódy řešení a adresu server, kde běží integrační server.

- webURL - adresa integračního serveru
- trunkUrl - umístění zdrojových kódů

Konfigurace sestavení

Konfigurace sestavení probíhá v souboru **build.build** (příklad souboru je uveden v příloze A .2), který musí být součástí repositáře řešení, které chceme sestavit na integračním serveru. Parametry konfiguračního souboru jsou dokumentované a jména proměnných jsou dostatečně vysvětlující. Jak již ale bylo zmíněno v úvodu této části, sestavení využívá spoustu specifických nastavení a programů, které bohužel nemohou být součástí přílohy této práce.

Hlavní parametry konfigurace jsou:

- ProjectName.CI - jméno projektu, které je použito dále pro nasazení, jména adresářů a další
- Working.dir - pracovní adresář, který je automaticky nastaven na umístění skriptu **build.build**
- Artifacts.dir - adresář výsledku sestavení nastaven na proměnnou integračního serveru nebo na adresář CCNetSandBox

- SolutionPath - jména a relativní umístění vzhledem k pracovnímu adresáři k souborům řešení - pokud je nutno sestavit více řešení v jednom běhu, je možné oddělit tyto středníkem
- TaskServer.sln - specifické nastavení pro službu řešení nad platformou DevExpress

Další nastavení se týkají adresářů, do kterých se mají jednotlivé artefakty umístit, dle jejich typu (komprimovaná soubory na FTP, soubory pro aktualizace lokálních instancí na síťové úložiště a podobné).

Ostatní nastavení jsou již specifická vzhledem k platformě DevExpress a pro základní sestavení nejsou nutná. Voláním *nant clean compile* by mělo být možno definované řešení sestavit lokálně. Nastavením těchto NAnt úloh v konfiguraci úlohy (sekce *tasks - nant - targetList - target* v souboru konfigurace ukázaném v příloze A .1) na integračním serveru bude totéž vykonáno i na integračním serveru v případě změny řešení (commitu do SVN).

Výsledky sestavení na integračním se serveru budou prezentovány v rámci webového rozhraní.

Index